Санкт-Петербургский государственный университет

Кафедра технологии программирования

Семёнов Сергей Владимирович

Выпускная квалификационная работа бакалавра

Формирование виртуального видеопотока на основе данных о движущихся объектах

Направление 010400

Прикладная математика и информатика

Научный руководитель,

ст. преподаватель,

Севрюков С.Ю.

Санкт-Петербург

2017

Содержание

Введение

Современный мир невозможно представить без информационных технологий, проникших во все сферы нашей жизни: тогда как первые вычислительные системы имели огромный размер и были применимы только для численного решения научных задач, сегодня их миниатюрные аналоги широко используются в быту и сопровождают нас повсеместно.

После создания в 70-ых годах XX-го века первых цифровых камер, а также с учётом интенсивного роста вычислительных мощностей компьютерные вычисления стали активно применяться для цифровой обработки изображений, что в итоге привело к необходимости решения задачи распознавания изображений при помощи компьютерного зрения. С каждым днём количество задач, которые решаются с его помощью, неуклонно растёт, и его технологии используются всё чаще ввиду удешевления и повышения доступности не только устройств фото- и видеозаписи (например, оснащение смартфонов камерами высокого разрешения), что послужило причиной небывалому расширению базы изображений и видео, но и устройств хранения и обработки данных.

Компьютерное зрение также применяется для обработки видеопотоков, которые, хотя и могут быть представлены как последовательность изображений, из-за чего можно применять к ним все используемые при работе с изображениями алгоритмы, в то же время имеют дополнительные присущие им свойства. Это позволяет решать не только те же задачи, что и для изображений, с использованием более эффективных учитывающих эти свойства алгоритмов, но и некоторые новые. Например, помимо обнаружения и классификации объектов компьютерное зрение позволяет также отслеживать эти объекты, для чего используется свойство связности изображений в видеопотоке.

Так как человеку для анализа цифровой информации может потребоваться немалое количество времени, а её объём неуклонно растёт, возникает необходимость в способе обработки и представления данных таким образом, чтобы повысить скорость их визуального восприятия. Если эти данные представлены в виде коллекции видео, на которых присутствуют мешающие восприятию движущиеся объекты, для этого можно использовать специальную систему, основанную на машинном обучении и компьютерном зрении, которая будет автоматически переключаться между элементами коллекции.

В данной работе будут рассмотрены существующие наработки, которые могут приблизить к решению данной проблемы, а также будет предложен один из способов построения такой системы. Конечным результатом работы является прототип программного обеспечения.

Цель, постановка задачи, возможности использования

Определения и понятия:

Под кадром в данной работе будем понимать изображение, полученное с видео в фиксированный момент времени.

Таким образом, **видеопоток** будем рассматривать как последовательность изображений (или кадров) с дискретным временем.

Под **оператором** подразумевается субъект, управляющий совокупностью аппаратных средств, получающих видеоданные и записывающих их на физическое устройство хранения данных. Каждый оператор характеризуется своим положением в пространстве.

Назовём **сценой** совокупность связанных своим местоположением объектов. Предполагаем, что сцена содержит участок, представляющий для пользователя особый интерес - **область интереса** сцены.

Информативным будем называть кадр, с помощью которого можно получить *представление о сцене*, и, что важнее, требуемое *представление об области интереса сцены*.

Тогда наиболее информативный кадр из какой-то коллекции видео - кадр, максимально точно описывающий сцену в фиксированный момент времени, то есть из которого можно получить наилучшее *представление об области интереса сцены* в данный момент времени.

Будем называть **представлением видеоданных** способ организации цифровой информации, повышающий эффективность их визуального восприятия: чем меньше время, требующееся человеку на визуальный анализ информации, тем лучше представление.

Наряду с кадром, представление видеоданных также обладает свойством *информативности*: наиболее информативное представление будет описывать сцену максимально точно.

Цель работы:

Основная **цель** данной работы - повышение информативности и эффективности представления видеоданных.

В данной работе выдвигается **гипотеза** о том, что существуют методы, позволяющие выявить данные, использование которых поможет совместить кадры из разных видео (то есть создать виртуальный видеопоток), что поможет в достижении поставленной цели.

Создание виртуального видеопотока повысит информативность представления цифровой информации и позволит сократить затрачиваемое на визуальный анализ время. Также предлагается отслеживать движущиеся объекты, мешающие видимости операторами области интереса сцены. Это, в свою очередь, поможет определить участок виртуального видеопотока, в который следует осуществить переход, что позволит ещё больше повысить информативность данного представления.

Постановка задачи и область применения:

Имеем коллекцию видео, которые могут иметь отличную друг от друга частоту кадров в секунду и не совпадать по размеру кадра. Каждое видео из коллекции получено от оператора. Разные операторы имеют различное местоположение и угол съёмки. Предполагается, что все получаемые операторами видео принадлежат одной общей сцене.

Сцена содержит движущийся объект или объекты, в определённый момент времени перекрывающие (полностью или частично) видимость области интереса каким-либо оператором или операторами. Ставится задача разработки такого прототипа программы, который должен автоматически отслеживать такие объекты и переходить, не дожидаясь перекрытия, к тому участку виртуального видеопотока, который будет давать максимальную

информативность. Это значит, что он должен давать максимально точное представление об области интереса сцены в нескольких последовательных моментах времени, следующих за перекрытием текущего участка виртуального видеопотока движущимся объектом (обозначим эту последовательность за М). Таким участком может быть как положение какого-либо оператора, и тогда виртуальный видеопоток в моменты времени М будет представлять собой видео, получаемое определённым оператором в эти моменты времени, так и изображение, полученное синтезированием общего кадра из нескольких кадров, получаемых различными операторами в данные моменты времени.

В этих условиях ставится **задача**: подтвердить или опровергнуть выдвинутую гипотезу. Проверять гипотезу предлагается при помощи **эксперимента**, состоящем в использовании определённых алгоритмов. В случае подтверждения гипотезы в результате эксперимента будет получено готовое **решение**, представленное в виде прототипа программы.

Область применения такой программы крайне обширна и неполный список включает, в частности, следующие сферы:

- Спортивные события.
- Телемедицина.
- Политические дебаты.
- Массовые мероприятия.
- Общественные места.
- Наблюдение за рабочим процессом на предприятиях.
- Охрана важных объектов.
- В магазинах и на заправках.

Рассматриваемая система может быть интегрирована как в онлайновые, так и в офлайновые сервисы и системы. Её части можно применять в любых задачах, где требуется:

- Находить области движения.
- Находить движущиеся объекты.
- Предсказывать траекторию движущихся объектов.
- Осуществлять сравнение изображений, а также среди коллекции видео находить то, которому принадлежит кадр.

Глава 1. Определение требований и выбор подхода

В этой главе будут рассмотрены имеющиеся решения задач построения виртуального видеопотока и поиска движущихся объектов. Также будут представлены возможные подходы к решению поставленной задачи и на основании имеющихся требований и ограничений будет выбран один из них.

1.1. Исследование существующих решений

Решение поставленной задачи должно включать не только технологии *построения виртуального видеопотока*, но и технологии *видеоаналитики* (чаще всего применяемой в задачах видеонаблюдения), позволяющей проводить анализ видео с использованием компьютерного зрения. В общем случае для каждой из них можно выделить два подхода: программный и аппаратный.

Программным подходом является некоторая совокупность алгоритмов, на вход которой подаются входные данные, после чего происходит их автоматическая обработка и возвращение требуемого результата.

Тогда как **аппаратный подход** - это система, включающая как программную часть, так и специальное оборудование, использующееся для записи, хранения и передачи данных. Программная часть в данном случае также служит для координации работы каждого отдельного элемента системы.

Сравнение подходов

К преимуществам второго подхода перед первым следует отнести не только высокое качество работы, но и быстродействие. Это достигается при помощи использования специального оборудования, которое позволяет как

получать входные данные высокого качества, так и использовать большую вычислительную мощность для их обработки. Более того, в таком случае программная часть также оптимизирована под конкретные элементы системы, что и служит главной причиной высокой скорости работы. Главными же недостатками данного подхода являются крайне высокая стоимость необходимого оборудования и специальное требование: нужно иметь специалиста, который будет осуществлять первоначальную настройку и работу такой системы. Кроме этого, она не будет функционировать с использованием каких-либо посторонних аппаратных элементов, из чего следует невозможность быстрой и дешёвой замены элемента, вышедшего из строя.

Программный же подход не требует дорогостоящего оборудования, но в то же время очень сильно зависит от качества входных данных и конфигурации вычислительной системы, которая будет осуществлять требуемые вычисления. Более того, в таком случае необходимы дополнительные этапы обработки изображений и алгоритмы, способные показывать хорошие результаты на любой вычислительной системе и при любом качестве входных данных.

Видеоаналитика для видеонаблюдения

Одним из представителей аппаратного подхода к интеллектуальному видеонаблюдению является система Kipod [1] от компании Synesis для интеллектуального видеонаблюдения, позволяющая максимизировать производительность труда оператора, осуществляющего видеонаблюдение, за счёт использования видеоаналитики (Рис. 1).

Рисунок 1. Упрощённая схема системы видеонаблюдения Kipod и Kipod Server (http://synesis.ru/assets/images/Produkti/kipod/systema-kipod-shema.jpg, http://synesis.ru/assets/images/Produkti/kipod/servOnvif(1).jpg)

Система может автоматически переключать камеру при переходе движущегося объекта из зоны видимости одной камеры в зону видимости другой, а также предоставляет широкий спектр возможностей удобной настройки правил видеоаналитики. Самая минимальная стоимость только для этих целей на январь 2017 года, не считая стоимости камер и установки, составляет около 500 тысяч рублей.

Системы построения виртуального видеопотока

Представителям аппаратного подхода к построению виртуального видеопотока является Sony's 4K Camera Stitching Technology [2]. Программное обеспечение системы позволяет объединить получаемые с помощью двух 4K камер PMW-F55 изображения в единое с последующим выделением любой его части для дальнейшего детального наблюдения. Также, так как положение камер установлено и известно заранее, система позволяет производить видеоаналитику (отслеживание игроков, выделение области, в которой происходит активная игра и т.д.) без высоких вычислительных затрат. Она была успешно испытана на Кубке

Конфедераций 2013, проходившем в Бразилии, совместно с FIFA TV (Рис. 2).

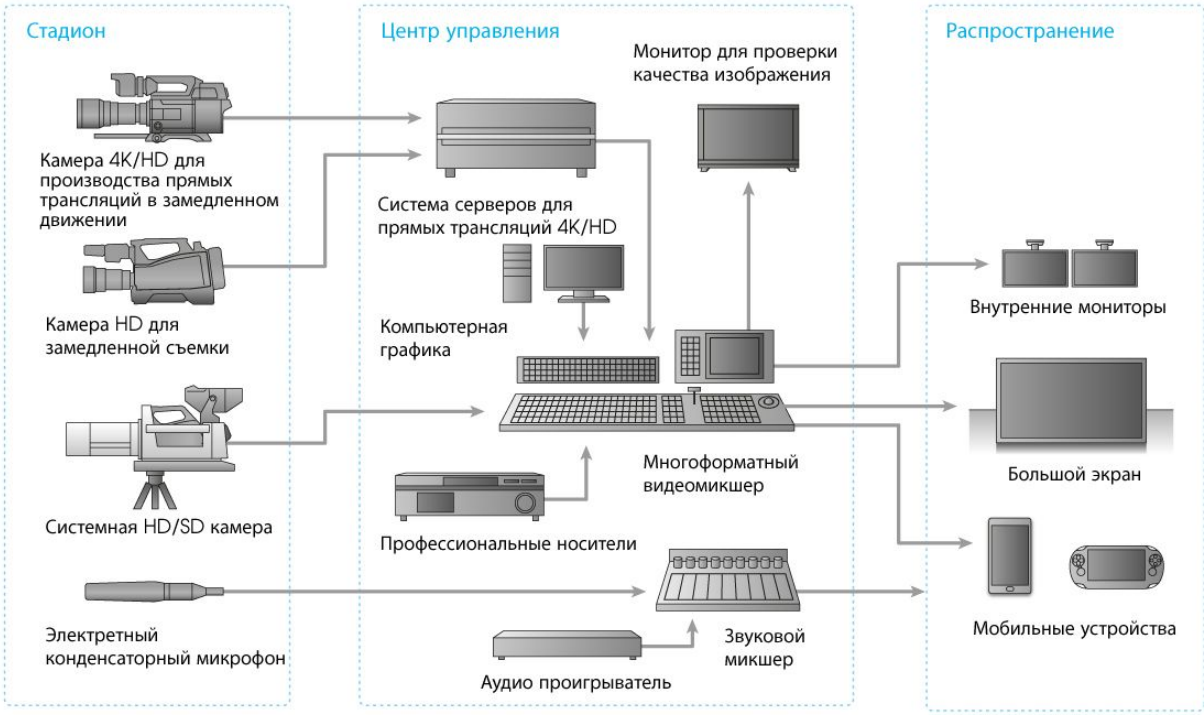

Рисунок 2. Общая схема системы Sony (http://assets.pro.sony.eu/Web/commons/solutions/images/ru/environment/sports-stadiums/hd-live-production-systems/system-configuration-example/image_1.jpg)

Согласно полученной из открытых источников информации [3, 4], на апрель 2016 года средняя стоимость оборудования такой системы составляет 15 миллионов рублей. Хотя она и обеспечивает работу с 4K видео в режиме реального времени с более чем хорошим качеством, её невозможно использовать во многих ситуациях ввиду ограничений на денежные ресурсы.

Одной из систем построения виртуального видеопотока, основанной на программном подходе, является LensFrame [5] от компании Coherent Synchro. Она позволяет посредством объединения нескольких видео, записываемых независимо друг от друга с HD камер, получить единое с возможностью последующего приближения для фокусировки на каком-либо объекте (Рис. 3). Кроме того, существует возможность управлять виртуальной камерой и интегрировать систему 3D визуализации. Минусом этого программного

обеспечения является необходимость ручной настройки положения видеопотоков относительно друг друга. Также стоит отметить, что, судя по активности на сайте компании, продукт больше не развивается (последняя новость датируется 26.02.2014).

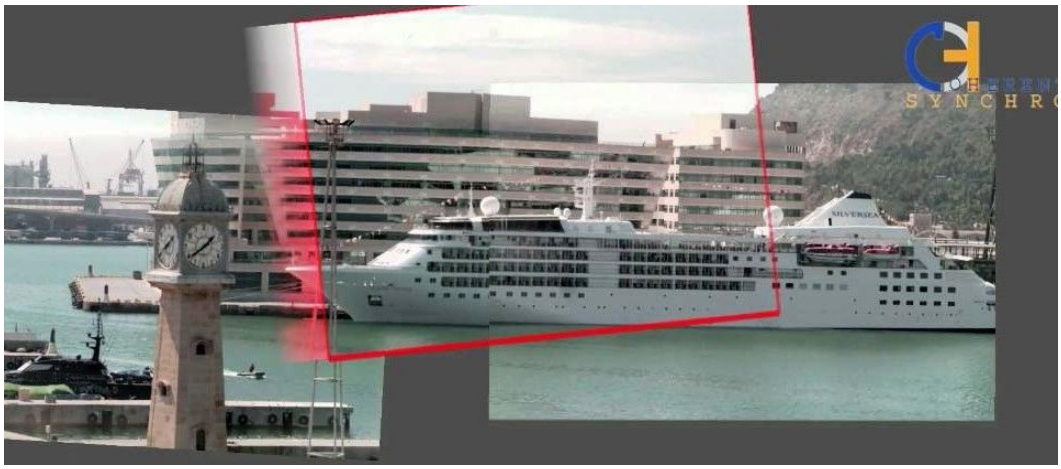

Рисунок 3. Ручная настройка положения видеопотоков относительно друг друга (http://www.coherentsynchro.com/contingut/uploads/Seaport_Stitching-960x417.jpg)

Вывод

Таким образом, существующие системы либо предъявляют жёсткие требования к аппаратной составляющей, либо вовсе не учитывают движения объектов. С учётом этого, а также ввиду того, что программный подход является более универсальным, нежели аппаратный (программную систему можно расширить до аппаратной, добавив поддержку оборудования и получив таким образом полную программную составляющую аппаратной системы), в рамках данной работы сделан упор на программную реализацию построения виртуального видеопотока с использованием видеоаналитики.

1.2. Ограничения и требования

В данном параграфе будут установлены требования к входным данным и их ограничения, а также требования к искомому решению.

Требования к операторам и их ограничения

Все операторы должны быть расположены таким образом, чтобы на каждом из получаемых ими видео присутствовала область интереса (Рис. 4). Кроме этого для каждого оператора должна быть известна область на изображении, которая представляет область интереса, и его приоритет: чем больше приоритет, тем более информативные с точки зрения пользователя он получает кадры (при условии, что область интереса не перекрыта объектом). Местоположение операторов, как и их взаимное расположение, знать не требуется.

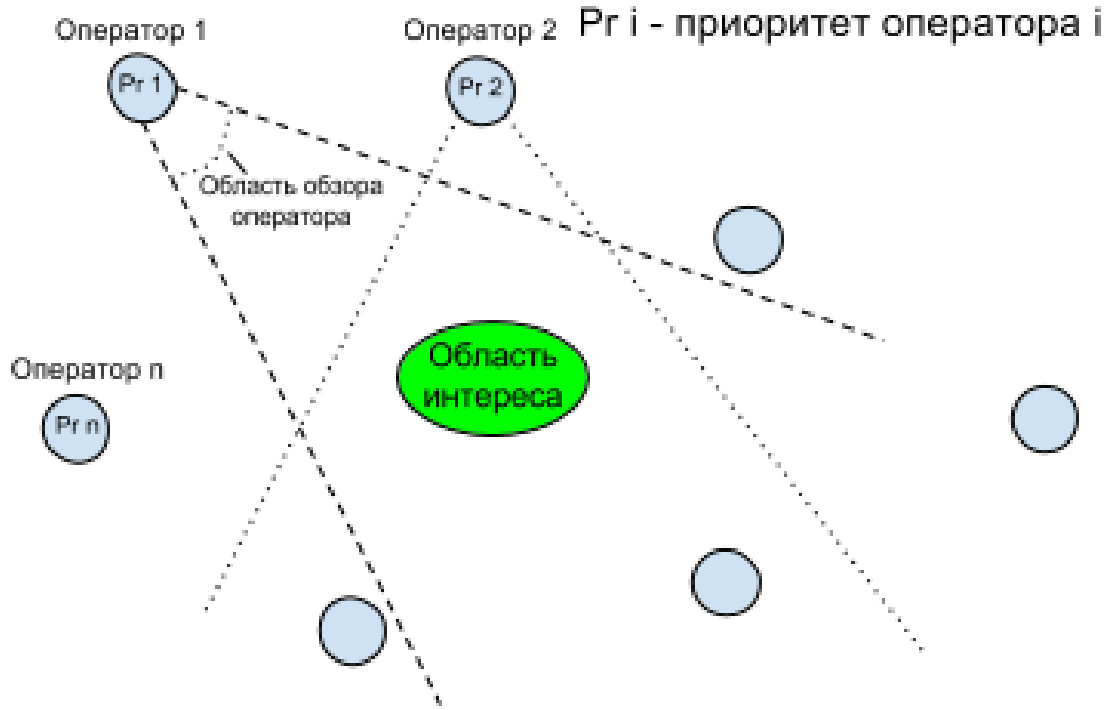

Рисунок 4. Расположение операторов и их приоритет

Требования к входным видео

Все получаемые операторами видео должны быть приемлемого качества. Это включает как разрешение изображений (качество обработки изображений, которые были подвергнуты очень сильной компрессии, будет падать. Высокое же разрешение входных изображений повысит качество

работы программы, но скажется на быстродействии), так и возможность выделить объекты (чем хуже освещение, тем сложнее их выделить. Также большое количество бликов и резкая смена освещения на изображении или его области сильно повлияет на качество их выделения).

Ещё одним требованием будет частичная синхронизация видео по времени: некоторый промежуток времени $t = [t_0, t_1]$, по которому будет строиться виртуальный видеопоток, у них должен быть общим:

$$t = \bigcup_{V_k \in V} time(V_k)$$

где V - множество входных видео, $time(V_k)$ - время, в которое было снято видео под номером k

Стоит отметить, что в рамках данной работы не требуется, чтобы видео были полностью синхронизированы по времени. Это означает следующее:

- У них может быть разное время начала съёмки.
- В случае, когда время начала съёмки совпадает, даже при одинаковой частоте кадров является допустимым, если $i-$ый кадр видео V_1 будет соответствовать кадру из видео V_2 под номером $j = i - k$, где k - некоторое целое число.

Требования к алгоритмам

Существует огромное множество различных методов работы с цифровыми изображениями, однако рассматриваемые в рамках данной работы должны удовлетворять некоторым требованиям, среди которых можно выделить:

- Высокая скорость работы.
- Устойчивость к шумам и искажениям.

- Устойчивость к вариации масштаба (в том числе обусловлено тем, что для каждого оператора расстояние от него до области интереса своё и не является фиксированным).
- Устойчивость к вращению оператора и изменению угла съёмки.

Под указанные требования хорошо подходят алгоритмы, основанные на локальных особенностях изображений [6]. В следующем параграфе будет выбран один из них.

1.3. Выбор метода нахождения и построения локальных особенностей

В качестве локальных особенностей в рамках данной работы будем рассматривать *особые точки*. **Особая (ключевая) точка / особенность** - обладающая рядом свойств точка, находящаяся на важном фрагменте изображения. Чаще всего за особые принимают некоторые *угловые* точки изображения, то есть точки, лежащие на пересечении двух и более *граней* на изображении.

Алгоритмы построения особых точек состоят из двух последовательных шагов:

- Определение положения ключевых точек с помощью *детектора*, обеспечивающего инвариантность нахождения одних и тех же особенностей относительно различных преобразований изображения.
- Получение числового вектора, описывающего особенность и её окрестность, называемого *дескриптором*.

Полученные дескрипторы можно сравнивать, используя различные метрики, что гораздо быстрее и даёт гораздо лучший результат, чем, например, попиксельное сравнение изображений. Также использование особых точек открывает огромное множество самых различных

дополнительных возможностей. Некоторые из них будут рассмотрены в рамках данной работы.

С учётом имеющихся требований среди алгоритмов построения особенностей необходим такой, который бы имел:

- Высокую *скорость работы*. Сюда включается как время, затрачиваемое алгоритмом на построение особых точек, так и время, которое требуется на их сравнение. Это особенно важно в задачах, требующих находить и сравнивать особенности в режиме реального времени.
- Высокое *качество* построенных особенностей. Сюда включается их инвариантность к повороту, изменениям масштаба, помехам, изменению освещения и разрешения изображения.

Наиболее быстрым алгоритмом на данный момент является Oriented FAST and Rotated BRIEF (ORB) [7], что позволяет использовать его в режиме реального времени даже при некоторых ограничениях на вычислительные ресурсы, при этом сохраняющим хорошие результаты. Ещё одним значительным преимуществом алгоритма является то, что он опубликован под BSD лицензией и с версии 2.3 присутствует в наиболее популярной библиотеке компьютерного зрения Open Source Computer Vision Library (OpenCV). Учитывая его преимущества и имеющиеся ограничения, в рамках данной работы для поиска ключевых точек был выбран именно этот алгоритм.

Детектор метода ORB основывается на алгоритме поиска особых точек Features from Accelerated Segment Test (FAST) [8]. Авторами ORB были внесены изменения в оригинальный FAST, чтобы добавить инвариантность к поворотам. Для того, чтобы оставить только самые хорошие особые точки, после этого ORB использует детектор углов Харриса, выбирая только те

угловые точки, которые были найдены и детектором углов, и улучшенным алгоритмом FAST.

В качестве метода построения дескрипторов ORB использует Binary Robust Independent Elementary Features (BRIEF) [9], который описывает каждую особенность бинарным вектором, состоящим из 256 элементов.

Глава 2. Сравнение изображений и видео

Так как в рамках данной работы не требуется, чтобы входные видеопотоки были полностью синхронизированы по времени, появляется необходимость среди всех видео находить тот кадр, в который следует осуществить переход при перекрывании движущимся объектом области интереса. Общая схема этого процесса будет представлена в главе 5. В данной же главе будет рассмотрено, каким образом происходит описание каждого кадра, их сравнение и нахождение тех видео, которым может принадлежать определённый кадр.

2.1. Описание кадров.

Сравнивать изображения можно самыми разными способами: с помощью перцептивного хэша, цветовых гистограмм, матриц переходов, прямого сравнения особых точек и даже попиксельно. Однако чтобы, имея некоторое время на подготовку, в дальнейшем можно было быстро и качественно провести сравнение изображений, а также с целью дальнейшего обучения классификатора для поиска видео по изображению предлагается использовать подход под названием “Мешок визуальных слов” (BoW) [10]. Он может быть описан следующей последовательностью шагов:

1. Для каждого видео из набора входных видео V извлекаются особые точки по всем кадрам.
2. Все полученные дескрипторы кластеризуются. Центроидом каждого кластера будет являться “визуальное слово”. Все такие центроиды будут составлять словарь.
3. С помощью полученного словаря и имеющихся дескрипторов для каждого кадра строится дескриптор: гистограмма частот встречаемости “визуальных слов” на данном кадре (Рис. 5).

4. Для каждого кадра известен его дескриптор и класс (видео, которому он принадлежит). Можно обучить классификатор находить видео по гистограмме частот изображения.

Рисунок 5. Гистограмма частот встречаемости “визуальных слов” на изображении

После применения данного метода будет получена некоторая совокупность технических данных, которая позволит не только проводить быстрое сравнение изображений и осуществлять поиск видео по кадру, но и сократить время работы для некоторых подходов к выделению движущихся объектов.

2.2. Составление словаря “визуальных слов”

После того, как для каждого кадра каждого видео найдены особые точки, полученное множество особых точек необходимо кластеризовать, чтобы получить “визуальные слова”, которые будут являться центроидами полученных кластеров. Стандартный подход к кластеризации дескрипторов - использование широко используемого метода кластеризации k-means [11, 12]. ORB строит бинарные дескрипторы, поэтому для сравнения двух таких

дескрипторов в качестве метрики необходимо использовать расстояние Хэмминга:

$$Hamming\ distance = \sum_{x_i \neq y_i} 1,\ i = 1,\ \ldots,\ n$$

где x, y - бинарные векторы, n - их длина

Однако для k-means, реализованного в библиотеке OpenCV, не предусмотрен выбор метрики: для определения расстояния между векторами используется только Евклидова норма. Более того, даже если реализовать собственный k-means, использующий в качестве метрики расстояние Хэмминга, то он всё равно не будет подходить для бинарных векторов: k-means стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров. В случае с вещественными векторами центроиды кластеров можно определить как центр масс принадлежащих кластеру элементов, однако такой подход неприменим к бинарным векторам, ведь если вычислить их центр масс, то будет получен не бинарный, а вещественный вектор, не являющийся центроидом кластера (Рис. 6). Как тогда осуществить кластеризацию для получения “визуальных слов”?

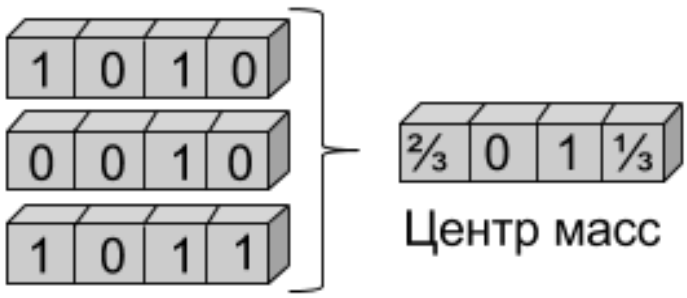

Рисунок 6. Попытка найти центроид бинарного вектора стандартным способом

Указанную проблему решает алгоритм кластеризации, получивший название k-majority [13]. Он был разработан с целью использовать идеи k-means там, где последний использовать невозможно, в том числе при работе с бинарными дескрипторами. На его создание авторов побудило именно желание интегрировать дескрипторы ORB в модель “Мешок

визуальных слов”, в то же время сократив время, которое требуется на кластеризацию. В основе алгоритма лежат идеи голосования и использования расстояния Хэмминга. Метод можно описать следующей последовательностью шагов:

1. Случайным образом генерируется k бинарных центроидов.
2. Для каждого вектора из множества дескрипторов, которое необходимо кластеризовать, с помощью расстояния Хэмминга ищется ближайший к нему центроид.
3. Для каждого кластера выбираются дескрипторы, ближайшим центроидом которых оказался центроид данного кластера. Каждый такой вектор голосует за каждый бит нового центроида следующим образом: к вектору-аккумулятору, каждый элемент которого изначально равен нулю, прибавляется единица, если соответствующий бит дескриптора - единица.
4. При помощи “правила большинства”, благодаря которому алгоритм получил своё название, выбирается каждый бит нового центроида: если большинство векторов проголосовало за единицу, бит принимает значение 1, иначе - 0.
5. Если хотя бы один центроид изменился относительно прежнего, возвращаемся к шагу 2. Если же новые центроиды получились теми же, искомые центры кластеров найдены.

2.3. Классификатор

Если составлен словарь “визуальных слов” и с его помощью для каждого кадра получена гистограмма частот, то имеются тренировочные данные, на основании которых можно обучить классификатор по гистограмме частот произвольного кадра определять похожее видео из коллекции. Существует множество различных классификаторов, основанных

на самых разнообразных подходах, и вопрос о том, в какой задаче какой из них следует использовать, является далеко не самым тривиальным. В этой подглаве приведён анализ существующих подходов и среди них выбран тот, который наиболее подходит в условиях имеющихся ограничений.

2.3.1. Метод опорных векторов

В качестве такого классификатора наиболее часто используется метод опорных векторов (Support Vector Machine, SVM) [14]. Ключевой принцип его работы - это переход от пространства, в котором заданы векторы обучающей выборки, к пространству более высокой размерности с помощью специальной функции, называемой ядром. После этого в полученном пространстве осуществляется поиск оптимальной разделяющей гиперплоскости, то есть расстояние от которой до ближайших элементов классов будет максимальным. Опорными векторами называются векторы, лежащие ближе всех к разделяющей гиперплоскости (Рис. 7).

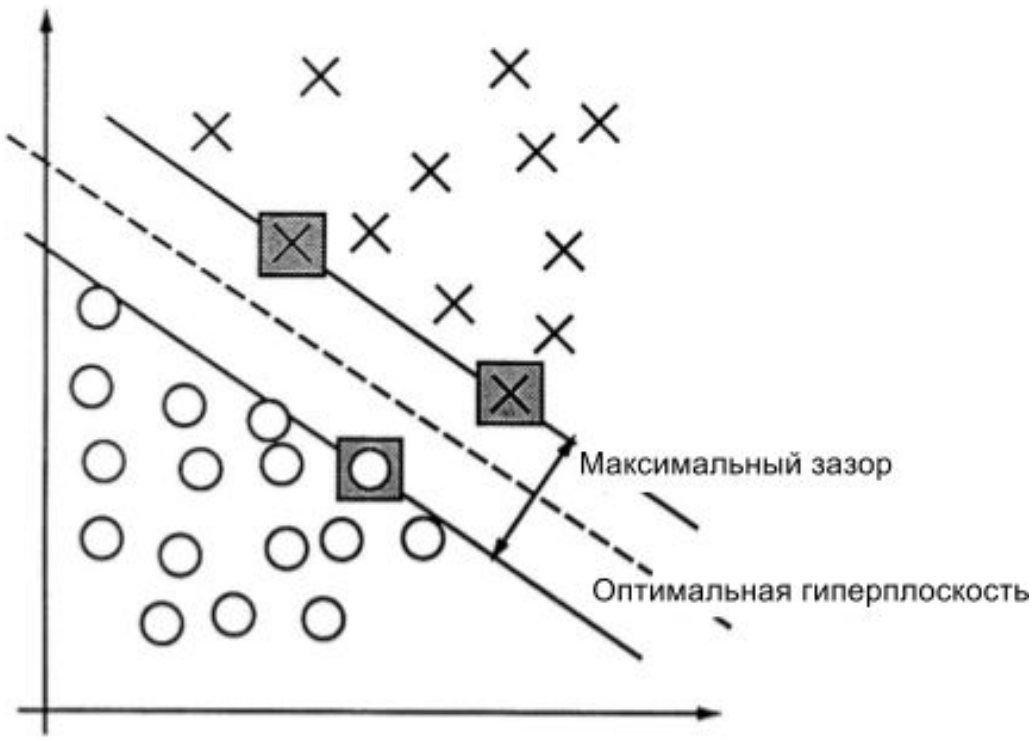

Рисунок 7. Иллюстрация метода опорных векторов [14]

Так как в данной работе SVM имеет дело с векторами, представляющими собой гистограммы частот, в качестве ядра для достижения более высокой точности классификации целесообразно использовать ядро хи-квадрат [15], имеющее следующий вид:

$$K(x,y) = exp\,(-\gamma \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i}) \quad (1)$$

где x, y - гистограммы частот, n - их длина, $\gamma > 0$

К сожалению, в рамках данной работы при использовании стандартного SVM могут возникнуть сложности. Если входной кадр, который нужно классифицировать, будет принадлежать какому-либо видео из коллекции, то, скорее всего, классификатор в качестве похожего видео вернёт именно его. В рамках данной работы же, как было сказано ранее, необходимо не узнать, какому из N видео принадлежит данный кадр, а найти ему похожее. В таком случае можно возвращать не первый, а, например, второй по результату классификации класс, который может быть найден при помощи распределения вероятностей.

Таким образом, необходимо получить вектор $P = (p_1, \ldots, p_N)$, каждый элемент которого характеризует вероятность принадлежности изображения определённому классу (так называемая “мягкая” классификация). Тогда класс, соответствующий второму по величине элементу вектора, и будет наиболее похожим видео на то, кадром которого является входное изображение.

SVM изначально не является вероятностной моделью классификации, и, хотя существует множество способов свести её к таковой [16], эти имплементации имеют существенные недостатки. Например, наиболее популярный из способов, получивший название Шкалирование по Платту (Platt scaling), имеет некоторые проблемы теоретического плана и во многих случаях оценивает распределение неверно. Наиболее же значительным

недостатком рассматриваемого классификатора для данной работы является увеличение времени классификации при переходе к вероятностной модели, что негативно отразится на быстродействии программы в режиме реального времени.

2.3.2. Метод релевантных векторов

Актуальной альтернативой методу опорных векторов является так называемый метод релевантных векторов (Relevance Vector Machine, RVM) [17], который активно применяется и демонстрирует хорошие результаты в различных задачах классификации [18, 19]. Данный алгоритм, основанный на байесовском выводе, имеет две версии: 2001 года, запатентованная Microsoft, и его усовершенствованная версия 2003 года, находящаяся в свободном доступе. Первая, основывающаяся на методе максимального правдоподобия (EM) и методе простой итерации, сперва берёт все имеющиеся векторы и на каждой итерации исключает наименее подходящие на роль базисных (релевантных) векторов. Вторая же, наоборот, начинает работу с пустого множества базисных векторов, на каждом этапе добавляя к нему наиболее подходящие, в то же время контролируя, не нужно ли исключить какой-либо вектор. Помимо того, что второй метод находится в свободном доступе, он также значительно выигрывает у первого в быстродействии, поэтому именно он и использован в данной работе. Далее будут описаны основные идеи метода релевантных векторов.

На входе имеется набор тренировочных пар $\{x_i,\ t_i\}_{i=1}^{N}$. Предполагаем, что вектор $t = (t_1,\ \ldots,\ t_N)^T$ может быть представлен в следующем виде:

$$t = y + \varepsilon = \Phi w + \varepsilon$$

Здесь $\varepsilon = (\varepsilon_1,\ \ldots,\ \varepsilon_N)^T$ подчинено гауссовскому распределению $N(0, \sigma^2)$, $w = (w_1,\ \ldots,\ w_M)^T$ - веса, которые необходимо найти, $\Phi = [\phi_1,\ \ldots,\ \phi_M]_{NxM}$. Матрица Φ зависит от выбранного способа её

построения, и поэтому M может быть различным: равным как N, так и $N * M$. Например, иногда полагают, что $M = N + 1$, $\Phi_{n,0} = 1$, а также

$$\phi_i(x) = K(x, x_i)$$

где K - ядро, i = 1, ..., N

Далее вводится априорное распределение параметров w в виде:

$$p(w|\alpha) = (2\pi)^{-M/2} \prod_{m=1}^{M} \alpha_m^{1/2} exp(-\frac{\alpha_m w_m^2}{2})$$

$\alpha = (\alpha_1, ..., \alpha_M)^T$ - независимые гиперпараметры регуляризации, каждый из которых отвечает за свой вес из w. Формула Байеса примет вид:

$$p(w|t, \alpha, \sigma^2) = \frac{p(t|w,\sigma^2)p(w|\alpha)}{p(t|\alpha,\sigma^2)} = (2\pi)^{-(N+1)/2}|\Sigma|^{-1/2} exp(-\frac{1}{2}(w-\mu)^T\Sigma^{-1}(w-\mu))$$

Таким образом, получаем нормальное распределение $N(\mu, \Sigma)$ с параметрами

$$\Sigma = (\sigma^{-2}\Phi^T\Phi + A)^{-1},\ \mu = \sigma^{-2}\Sigma\Phi^T t$$

где $A = diag(\alpha_1, ..., \alpha_M)$

Для получения α решается задача максимизации так называемой маргинальной функции правдоподобия, имеющей следующий вид:

$$L(\alpha) = ln\ p(t|\alpha, \sigma^2) = -\frac{1}{2}[N ln(2\pi) + ln|C| + t^T C^{-1} t]$$

$$C = \sigma^2 I + \Phi A^{-1}\Phi^T$$

Для задачи классификации используются следующие формулы, полученные с использованием аппроксимации Лапласа:

$$\Sigma = (\Phi^T B\Phi + A)^{-1},\ \mu = \Sigma\Phi^T Bt,\ C = B^{-1} + \Phi A^{-1}\Phi^T$$

В этих уравнениях $B = diag(\beta_1, ..., \beta_M)$, $\beta_n = \sigma\{y(x_n)\}(1 - \sigma\{y(x_n)\})$, а σ - логистический сигмоид:

$$\sigma(y) = 1/(1 + exp(-y))$$

Таким образом, RVM изначально является вероятностным методом, и он не требует никаких дополнений для получения распределения вероятности принадлежности элемента определённому классу. Кроме того, отсутствует необходимость подбора параметра, отвечающего за влияние

входящих в модель векторов на решающее правило, так как автоматический подбор гиперпараметров является частью процесса обучения.

2.3.3. Сравнение методов опорных и релевантных векторов

Ключевым отличием RVM от SVM является использование гораздо меньшего числа векторов, задействованных в модели (опорных для SVM и релевантных для RVM), при помощи которых строится решающее правило (Рис. 8, 9), что ведёт к существенному приросту скорости предсказания класса [18].

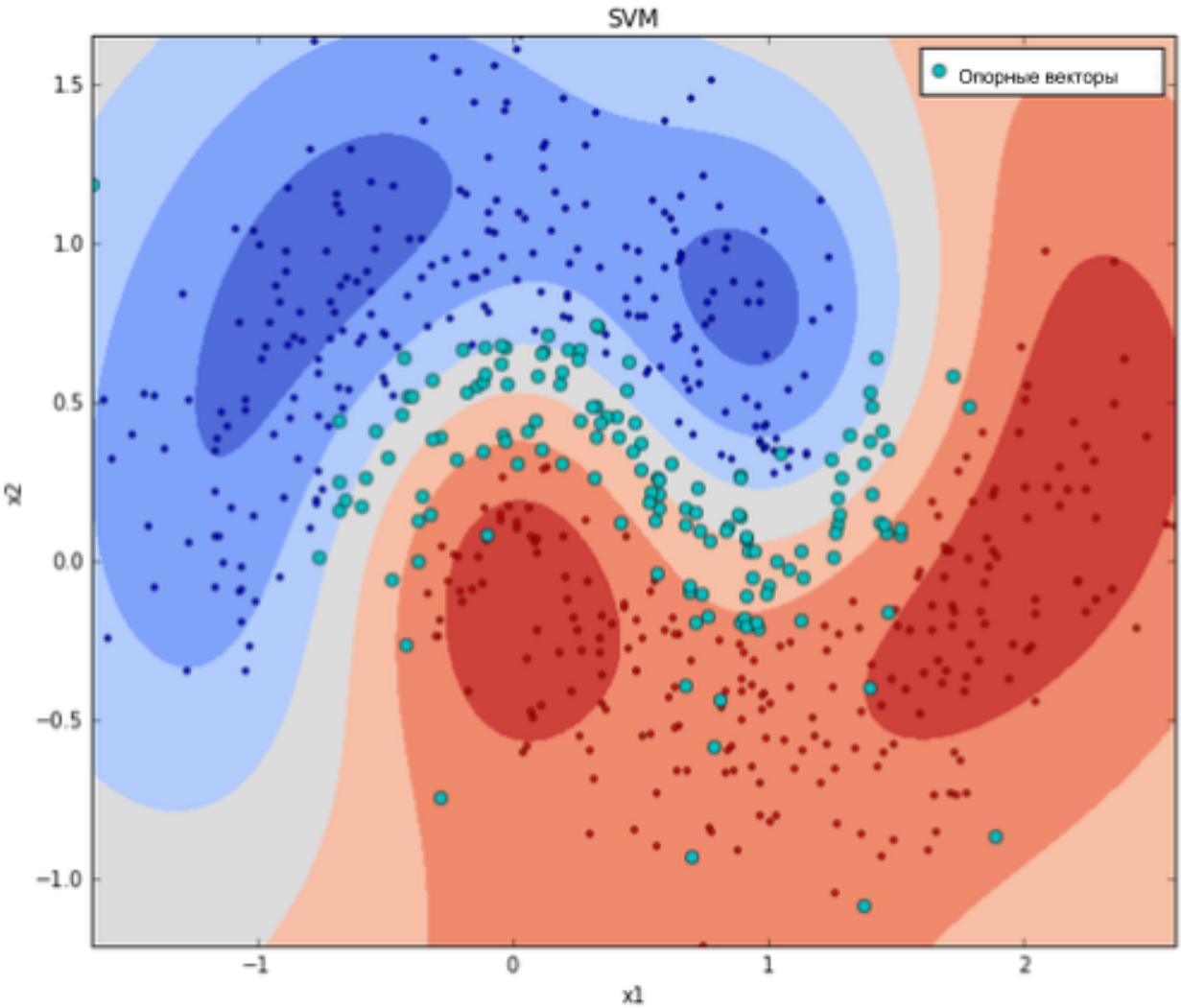

Рисунок 8. SVM. Выбор опорных векторов из тренировочного набора

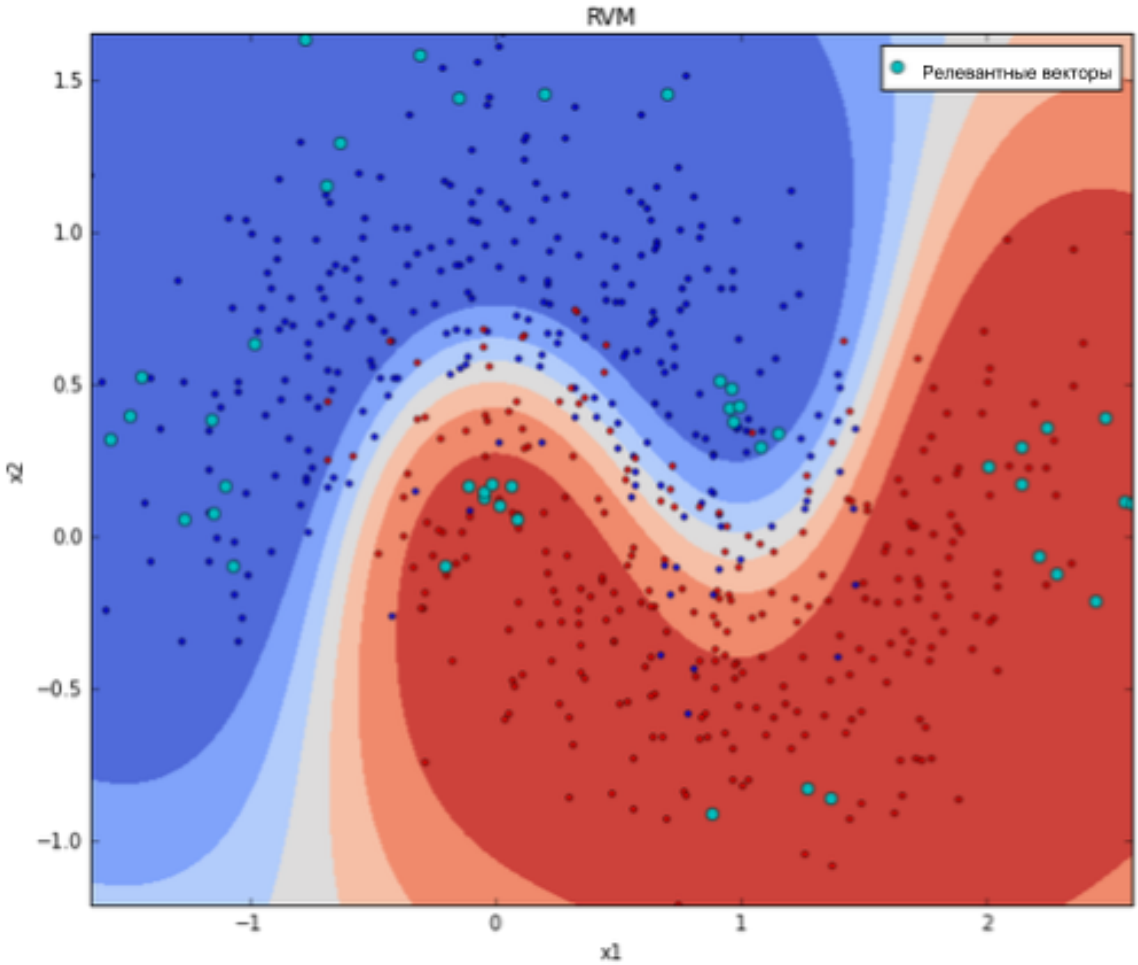

Рисунок 9. RVM. Выбор релевантных векторов из тренировочного набора

Как уже отмечалось выше, ещё одним преимуществом RVM перед SVM является отсутствие необходимости подбора параметра, который осуществляется либо вручную, либо с помощью крайне трудоёмкого скользящего контроля (cross-validation). Помимо этого метод релевантных векторов лучше ведёт себя в случаях, когда увеличивается количество классов, ровно как и когда уменьшается количество тренировочных данных. Однако любой алгоритм имеет определённые недостатки, и RVM - не исключение. На обучение у метода уходит гораздо больше времени, чем у SVM без использования скользящего контроля:

	SVM	RVM
Обучение	1.5754	7.9776
Классификация	0.00134	0.00091

Таблица 1. Сравнение времени, требующегося классификаторам (секунд)

В таблице 1 также приведены результаты сравнения времени, требующегося на классификацию (размер словаря - 200, изменение размера словаря как на скорости обучения, так и на скорости классификации сказывается незначительно). Хотя RVM работает быстрее, чем SVM, прирост оказался не таким значительным, каким ожидался, что может свидетельствовать о некоторых проблемах в реализации алгоритма.

Таким образом, RVM рационально использовать в тех случаях, когда требуется определить, к какому классу с какой вероятностью будет принадлежать входной элемент, в то время как быстродействие классификации значительно приоритетнее скорости обучения алгоритма.

Глава 3. Поиск областей движения

Важной задачей компьютерного зрения, решение которой необходимо для огромного множества прикладных задач, является детектирование движения в видеопотоке. Существует множество подходов к её решению, основывающихся на самых различных принципах, критериями выбора из которых является как информация о входных данных, так и цель, с которой осуществляется поиск движения.

Важнейшими критериями выбора того или иного подхода являются тип фона видеопотока и степень подвижности камеры. В зависимости от них выделяют три принципиально отличных друг от друга случая:

1. Камера неподвижна. Фон не меняется или меняется в допустимых пределах.
2. Камера подвижна. Фон практически не меняется.
3. Камера подвижна. Фон всё время меняется.

Первый случай является наиболее простым, так как движение можно определить как изменение фона. Во втором же случае, когда камера находится в движении, она сама служит источником изменения фона, даже если он статичен, что уже делает невозможным применение многих классических подходов, хорошо работающих для неподвижных камер. Наиболее же сложными являются ситуации, когда помимо самой камеры находится в движении и сам фон. Кроме того, для каждого из трёх перечисленных случаев дополнительную сложность создаёт движение в кадре нескольких объектов, в определённый момент времени перекрывающих друг друга.

Практически все задачи, связанные с определением движения в видеопотоках, могут быть разделены на три основные группы:

1. Поиск областей движения.

2. Поиск движущихся объектов.
3. Создание трёхмерной модели объекта по его изображениям.

Задачи второй группы представляют большую сложность, чем первой, так как помимо нахождения на изображении областей движения дополнительно требуют выделения движущихся объектов. Помимо этого, для многих прикладных задач необходимо осуществить анализ найденного движения, в связи с чем осуществляется построение траектории найденных объектов. Задачи третьей группы подразумевают поиск определённых свойств объекта по его последовательным изображениям для дальнейшей трёхмерной реконструкции.

Задача поиска движения, которая решается в данной работе, может быть отнесена ко второй группе (поиск движущихся объектов) и первому случаю (камера подвижна, фон не меняется).

Далее будет более детально рассмотрен подход, который был использован для этой работы, и описан каждый его этап.

3.1. Выбор метода поиска движущихся объектов

Осуществлять поиск движущихся объектов в видео можно различными способами, однако среди них необходимо выбрать такой, который бы не был чрезмерно затратным в плане вычислений и возвращал хорошие результаты, то есть наилучшим образом учитывал информацию о входных данных. Один из основных методов поиска движущихся объектов - это производить его в два этапа:

1. Находить области движения.
2. Каким-либо образом выделять в этих областях интересующие объекты, которые осуществляют движение.

Преимущество такого подхода состоит в том, что алгоритмы поиска областей движения обычно имеют относительно маленькую вычислительную

сложность. Таким образом, если недостаточно одних только областей движения для заключения, что это и есть интересующие движущиеся объекты, можно дополнительно применить более дорогие алгоритмы. Главным достоинством является то, что их придётся применять уже не ко всему изображению, а только к найденной области движения, что значительно снизит затрачиваемое время на вычисления. Случаи, когда искомым объектом нельзя назвать область движения, не предприняв дополнительных действий:

- Алгоритм поиска областей движения работает недостаточно хорошо для выбранного видео. Помимо движущихся объектов ввиду каких-либо обстоятельств за области движения были приняты фрагменты, соответствующие фону.
- Необходимо найти только какие-либо определённые движущиеся объекты, в то время как остальные не интересуют. Частным случаем является построение траектории движения интересующих объектов в ситуации, когда в кадре находится более одного объекта, осуществляющего движение.

Ещё одним достоинством такого подхода является то, что он открывает возможность использования алгоритмов, для которых необходимо предварительно задать область, ограничивающую движущийся объект.

Таким образом, если при имеющихся для задачи ограничениях на входные данные получается предварительно найти области движения, наилучшим решением будет являться использование именно этого подхода. В рамках данной работы рассматриваются только полученные с неподвижной камеры видео, фон которых либо статичен, либо меняется лишь незначительно. Кроме того, изначально может не иметься никакой информации о предполагаемом объекте движения, то есть может быть не

известно ничего из следующего списка, что могло бы помочь в его нахождении:

- Диапазон скорости, с которой будет осуществляться движение.
- Примерное начальное положение объекта.
- Тренировочный набор изображений.
- Размер.
- Цвет.

Учитывая отсутствие в общем случае каких-либо данных об искомых объектах, для решения задачи было решено предварительно найти области движения в кадре, то есть использовать описанный выше подход. Далее будет рассмотрен метод поиска таких областей, хорошо работающий для статичных камер.

3.2. Вычитание фона

Наиболее широко используемым методом предобработки изображений для их последующей обработки при условии, что камера статична, является вычитание фона (Background Subtraction, BS) [20, 21, 22, 23, 24]. В его основе лежит следующая идея: строить модель фона B (формула 2), которая будет включать в себя статичную часть видео, то есть всё, что может быть отнесено к фону, после чего вычитать её из текущего кадра, таким образом получая разницу между ними (Рис. 10).

$$B = \{b(x,y),\ 0 \le x \le width,\ 0 \le y \le height\} \quad (2)$$

где $b(x,y) \in [0,255]$*, width - ширина изображения, height - высота*

Кадр перед вычитанием обычно переводят в градации серого, поэтому каждый пиксель модели фона, которая может быть представлена, как изображение, будет характеризоваться интенсивностью серого цвета. Тогда вычитание производится следующим образом:

$$D(x,y) = |F(x,y) - B(x,y)|$$

где $F(x,y)$ *- текущий кадр*

Таким образом, после того, как будет произведено вычитание, будет получено изображение в градациях серого, каждый пиксель которого будет характеризовать движение.

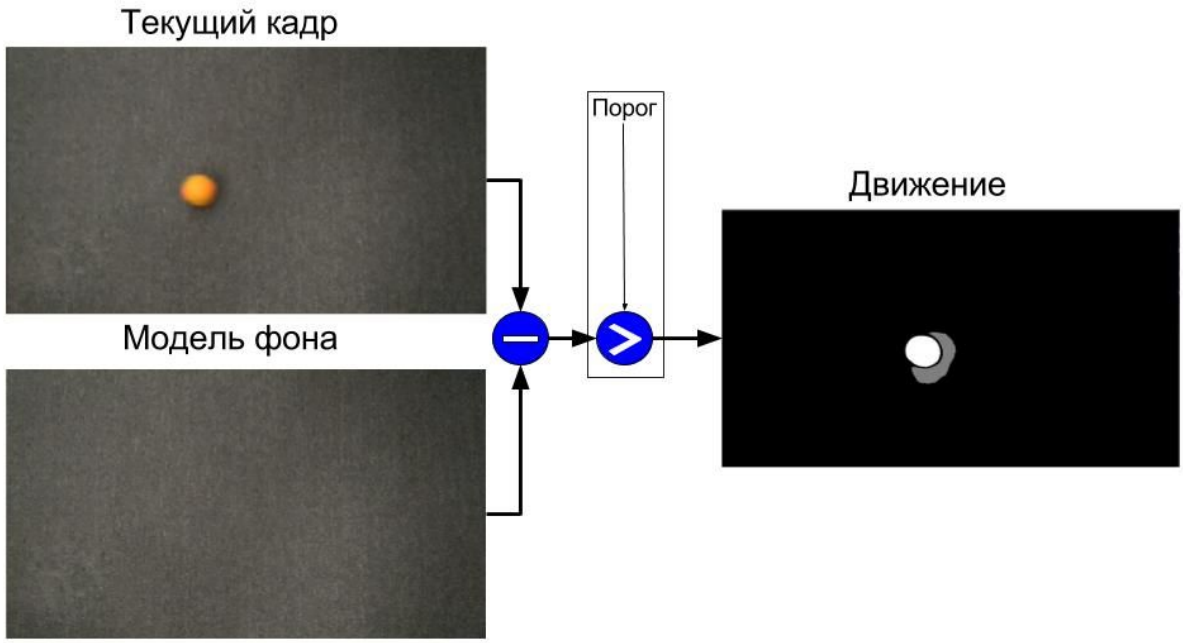

Рисунок 10. Вычитание фона

Чаще всего после получения изображения, характеризующего движение, к нему применяется бинарный порог с целью отобрать пиксели, принадлежащие движущимся объектам, от пикселей, которые относятся к фону (Рис. 11). Если подобрать правильный порог, у правого изображения на рисунке 10 не останется серой области, которая в данном случае соответствует тени движущегося объекта, то есть будет получено бинарное изображение (маску), белые области которого соответствуют движению, а чёрные - фону.

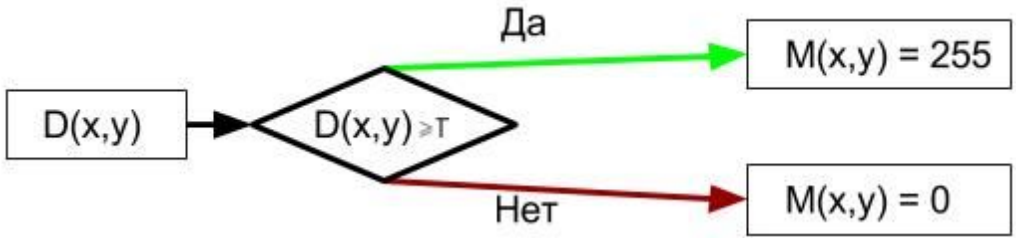

Рисунок 11. Бинарный порог. M - бинарное изображение (маска)

Из всего вышесказанного следует, что качество поиска движущихся объектов в данном случае очень сильно зависит от того, насколько хорошей была построена модель фона. Методы её построения делятся на две группы [20]:

- Нерекурсивные. Для обновления модели фона для текущего кадра используется интенсивность пикселей нескольких предыдущих моделей фона и текущего кадра.
- Рекурсивные. Для обновления модели фона для текущего кадра используется интенсивность пикселей только текущего кадра. Основное отличие от предыдущей группы - это адаптивность к фону и его последующим изменениям.

Далее будут рассмотрены некоторые подходы к составлению модели фона из первой и второй группы. Стоит отметить, что для рекурсивных методов процесс вычитания происходит несколько сложнее, чем на рисунке 10, что может быть замечено из приведённых далее алгоритмов, однако основная идея остаётся той же.

3.2.1. Нерекурсивные методы построения модели фона

Самым простым методом построения модели фона является метод вычитания текущего и предыдущего кадров. В качестве модели фона B для текущего кадра F_n он использует предыдущий кадр F_{n-1}, и тогда характеризующее движение для текущего кадра изображение D_n будет вычисляться следующим способом:

$$D_n(x,y) = |F_n(x,y) - F_{n-1}(x,y)|$$

Существует модификация данного метода, использующая для обновления модели фона не только предыдущий, но и следующий кадр [21]. Тогда правило трёх-кадровой разности для получения маски будет определяться следующей формулой:

$$M_n(x,y) = (|F_n(x,y) - F_{n-1}(x,y)| \geq T) * (|F_n(x,y) - F_{n+1}(x,y)| \geq T)$$

Также модель фона может быть получена с помощью использования нескольких последовательно идущих предыдущих кадров: например, путём их усреднения:

$$B_n(x,y) = \frac{1}{k} \sum_{i=1}^{k} F_{n-i}(x,y)$$

где k - количество используемых предыдущих кадров

Нерекурсивные методы легко реализовать, и, что важнее, они имеют высокую скорость работы, однако сильно зависят от скорости движения объектов: если она будет маленькой, то объекты будут плохо обнаруживаться. Кроме того, неудовлетворительный результат будет получаться как при изменении освещения (стоит отметить, что, хотя рекурсивные алгоритмы лучше справляются в такой ситуации, большинство из них всё равно не гарантируют приемлемого результата), так и в случаях, когда имеются незначительные изменения фона (например, присутствует листва деревьев или водоём в условиях ветра).

3.2.2. Рекурсивные методы построения модели фона. ViBe

Универсальным и наиболее совершенным на данный момент адаптивным методом, показывающим очень хорошие результаты практически для любых ситуаций и освещений, а также значительно выигрывающим в скорости работы по сравнению с другими алгоритмами,

показывающими приблизительно такое же качество поиска движущихся объектов, считается Visual Background Extractor (ViBe) [22].

Классическим принципом работы большинства активно использующихся современных адаптивных методов является построение для каждого пикселя кадра функции плотности вероятности. ViBe основывается на принципиально другой идее: к вопросу об интенсивности (в общем случае о цвете) пикселя $p = (x, y)$ подходят как к вопросу классификации. Для каждого пикселя сохраняется некоторое количество N его предыдущих значений $v(p)$, не обязательно последовательных (чаще всего полагают $N = 20$). Тогда можно сказать, что для каждого пикселя текущего кадра будет иметься своя модель C, которая представляется следующим множеством:

$$\mathrm{C}(p) = \{v_1(p), \ldots, v_N(p)\}$$

После этого осуществляется классификация пикселя с целью либо выявить движение, либо отнести данный пиксель к фону. В общем случае для этого в цветовом пространстве для пикселя p, значение которого на текущем кадре обозначим через $v_n(p)$, где n - номер текущего кадра, строится сфера $S_R(v_n(p))$ заранее определённого радиуса R (Рис. 12) и определяется количество K_n значений из $C_n(p)$, попадающих в эту сферу:

$$K_n(p) = |S_R(v_n(p)) \cap C_n(p)| \qquad (3)$$

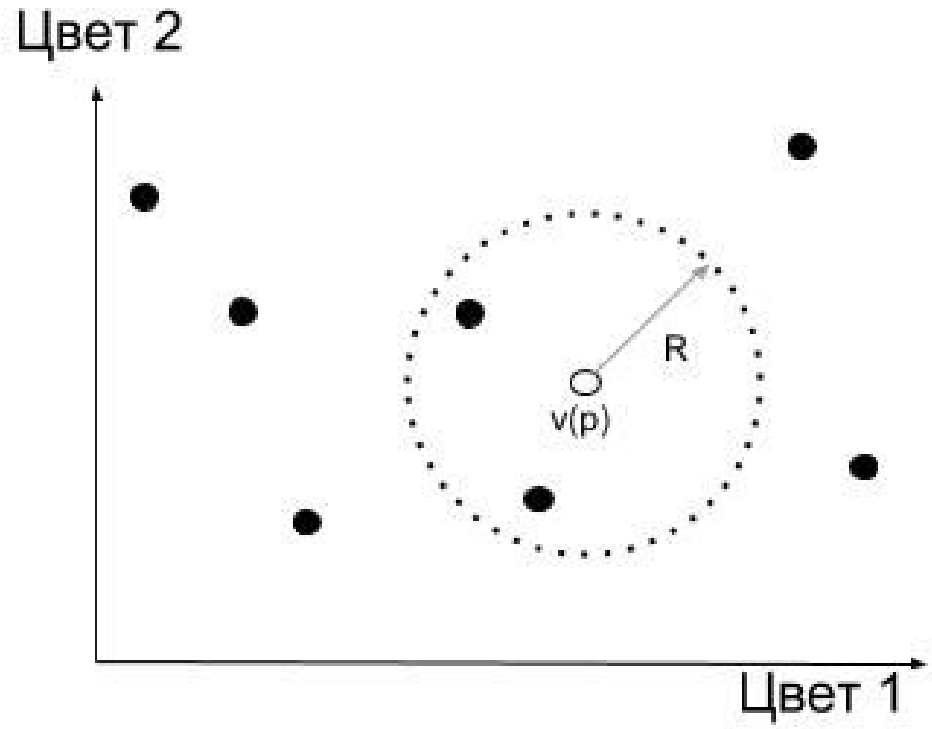

Рисунок 12. Сфера в цветовом пространстве

В случае, когда изображения предварительно переводятся в градации серого, значения $v(p)$ представляют характеризующий интенсивность серого цвета скаляр, а не вектор, и для вычисления количества K_n "близких" значению $v_n(p)$ элементов из $C_n(p)$ необходимо просто осуществить следующую проверку для каждого из данных элементов:

$$|v(p) - v_n(p)| < R \qquad (4)$$

Решающее правило для текущего кадра строится следующим образом: если $K_n > K_{min}$, где K_{min} обычно принимается много меньше, чем $N/2$, пиксель p на текущем кадре принимается за фоновый, иначе предполагают, что он принадлежит движущемуся объекту. Стоит отметить, что в формулах 3 и 4 можно осуществлять проверку не для всех для элементов $C_n(p)$, а только до тех пор, пока не найдём K_{min} попадающих в сферу (в общем случае).

После этого для каждого пикселя необходимо обновить его модель. Этот процесс представляет из себя два последовательных шага:

1. Если p был отнесён к фону, из $C_n(p)$ случайным образом выбирается элемент, который будет заменён на $v_n(p)$.
2. Из окрестности $O(p)$ размера 3x3 (9 пикселей с учётом p) случайным образом выбирается пиксель, случайный элемент модели которого также будет заменён на $v_n(p)$.

Кроме самого процесса обновления модели фона отличительной особенностью ViBe также является процесс её инициализации. Для многих алгоритмов она производится случайным образом, однако ViBe старается построить наиболее точную модель как можно раньше. С этой целью инициализация модели каждого пикселя производится так:

$$C_0(p) = \{v(z), z \in O(p)\}$$

Здесь пиксель z выбирается N раз случайным образом. Такой процесс позволяет получить достаточно хорошую модель фона уже для второго кадра видео.

Несмотря на все достоинства и простоту реализации, достаточно часто приходится искать альтернативу данному алгоритму, так как он защищен множеством патентов.

3.2.3. Рекурсивные методы построения модели фона. MOG

Стандартным подходом к построению модели фона, использующимся для многих прикладных задач, является смесь гауссовых распределений (Mixture of Gaussians, MOG) [23, 24]. Как упоминалось выше, чаще всего для каждого пикселя текущего кадра с номером n строится функция плотности вероятности $P_n = P(v_n(p))$, и MOG использует именно этот подход. Предполагается, что для каждого пикселя текущего изображения она может быть представлена смесью нормальных распределений, где G - их число в смеси. Обозначим за w_i^n вес распределения Гаусса с номером i, E_i^n - его

математическое ожидание и Σ_i^n - среднеквадратичное отклонение. Тогда для P_n получим:

$$P_n = \sum_{i=1}^{G} w_i^n * N(v_n(p)|E_i^n, \Sigma_i^n)$$

Здесь в общем случае $N(v_n(p)|E_i^n, \Sigma_i^n)$ - многомерное нормальное распределение, имеющее вид:

$$N(v_n(p)|E_i^n, \Sigma_i^n) = \frac{1}{(2\pi)^{H/2}|\Sigma_i^n|^{1/2}} exp[-\frac{1}{2}(v_n(p)) - E_i^n)^T * (\Sigma_i^n)^{-1} * (v_n(p)) - E_i^n)\ ] \quad (5)$$

где H *- число компонентов цвета,* Σ - матрица ковариации

Для ускорения вычислений, чтобы не вычислять обратную матрицу в формуле 5, в случае цветного изображения предполагается, что для компонентов цвета соблюдается их независимость, а также что они имеют одинаковое стандартное отклонение. Тогда матрица ковариации примет вид:

$$\Sigma_i^n = (\sigma_i^n) * E$$

где E *- единичная матрица размерности* H

После того, как для всех пикселей получена смесь, для каждой из них производится сортировка распределений по убыванию величины w_i^n/σ_i^n. Это делается для того, чтобы фоновые пиксели отвечали распределению с маленькой дисперсией и большим весом. Тогда число J первых гауссиан, соответствующих распределению цвета фона для пикселя p, будет определяться из условия:

$$J = arg\ min_a\ (\sum_{i=0}^{a} w_i^n > A)$$

где A *- некоторый порог*

Решающее правило для текущего кадра выглядит следующим образом: для каждого пикселя определяют, какому из распределений смеси принадлежит его значение $v_{n+1}(p)$, используя расстояние Махаланобиса:

$$\sqrt{(v_{n+1}(p)) - E_i^n)^T * (\Sigma_i^n)^{-1} * (v_{n+1}(p)) - E_i^n)} < 2.5 * \sigma_i^n$$

После этого возможно два случая:

1. Распределение нашлось. Пиксель будет отнесён к фону, если эта гауссиана является одной из J первых. В противном случае он будет отнесён к движущемуся объекту.
2. Если ни одной гауссианы не нашлось, то пиксель также относят к движущемуся объекту.

Теперь рассмотрим, как для пикселя p со значением $v_{n+1}(p)$ происходит обновление параметров его модели. Как и при принятии решения, относится ли пиксель к фону или нет, здесь также может быть два случая. Не существует единого понимания того, каким образом следует вычислять новые параметры, поэтому рассмотрим здесь один из наиболее популярных:

1. Распределение нашлось. Происходит обновление его параметров и веса. Если нашлось более одного распределения, данный процесс выполняется для каждого из них:

$$w_i^{n+1} = (1-\alpha) * w_i^n + \alpha$$

$$E_i^{n+1} = (1-g) * E_i^n + g * v_{n+1}(p)$$

$$(\sigma_i^{n+1})^2 = (1-g) * (\sigma_i^n)^2 + g * (v_{n+1}(p)) - E_i^{n+1}) * (v_{n+1}(p)) - E_i^{n+1})^T$$

$$g = \alpha * N(v_n(p)|E_i^n, \Sigma_i^n)$$

В этих формулах α *- заданная константа*

Для всех остальных распределений в смеси пересчитываются только веса:

$$w_i^{n+1} = (1-\alpha) * w_i^n$$

2. Распределение в смеси не нашлось. Тогда самую последнюю в уже отсортированной смеси гауссиану заменяют новой: $E_G^{n+1} = v_{n+1}(p)$, дисперсия - максимально возможная, а вес - минимально допустимый.

Для инициализации гауссиан для каждого пикселя чаще всего применяют либо EM-алгоритм (Expectation-maximization algorithm), либо k-means, что достаточно затратно в вычислительном плане. Число входящих

в смесь распределений G обычно принимают равным от 3 до 5. Также существует подход, позволяющий автоматически подбирать необходимое количество гауссиан [24].

3.3. Удаление шума

Не всегда простое применение алгоритмов вычитания фона к входному изображению позволяет приемлемым образом выделить движущиеся объекты, и одна из основных причин - различные присутствующие на нём шумы. Конечно, адаптивные алгоритмы вычитания фона пытаются с этим бороться, но далеко не во всех ситуациях это получается успешно, и тогда на помощь приходят различные фильтры обработки изображений [25]. Стоит помнить, что любые фильтры чувствительны к изменению параметров, и хорошо подобранные параметры для одной ситуации совсем не обязательно улучшат результат для другой. Далее будут рассмотрены фильтры, которые применялись в данной работе.

3.3.1. Размытие по Гауссу

Записывающие видео устройства никогда не бывают идеальными, поэтому даже при полном отсутствии движения не будут получены два одинаковых кадра: для двух одинаковых, как нам кажется, изображений некоторые пиксели обязательно будут иметь разные значения интенсивности цвета. Кроме того, на изображении возможны небольшие блики, поэтому перед обновлением модели фона для изображения его необходимо предварительно обработать. Одним из способов является размытие по Гауссу (Gaussian blur) (Рис. 13).

Рисунок 13. Исходное изображение (слева) и оно же после применения размытия по Гауссу с матрицей свёртки 9x9 (справа)

Размытие изображений предполагает использование ядра свёртки - матрицы коэффициентов, которые умножаются на значения интенсивности цвета пикселей изображения (Рис. 14). Операцию свёртки для каждого пикселя можно описать следующим образом:

1. Матрица свёртки “прикладывается” к пикселю, для которого вычисляется значение, элементом, называемым якорем. Обычно, как и на рисунке 14, им является центральный элемент матрицы, однако возможны и другие случаи.
2. Значение каждого пикселя, который попал под матрицу свёртки, умножается на соответствующий ему вес.
3. Полученные значения суммируются.

Для вычисления новых значений крайних пикселей изображения его обычно сначала расширяют до размеров, позволяющих “приложить” матрицу к данным пикселям. Это происходит либо с помощью простого дублирования этих пикселей, либо с помощью зеркального отражения пикселей относительно крайних.

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

Рисунок 14. Ядро свёртки с радиусом 1

Ядро свёртки, которое использует размытие по Гауссу, примечательно тем, что при вычислении весов оно учитывает взаимосвязь между пикселями, основываясь на расстоянии: чем ближе пиксель к якорю, тем больший ему соответствует вес. Для вычисления весов используется двумерное нормальное распределение, которое в данном случае примет вид:

$$G(x,y) = A * exp(\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2})$$

Где (x,y) - координаты элемента ядра, для каждой из координат σ *- среднеквадратичное отклонение,* μ *- центр гауссианы.* A *- коэффициент, вычисляемый с учётом* σ.

Координаты якоря полагают равными $(0,0)$, а среднеквадратичные отклонения либо задаются вручную, либо вычисляются, исходя из радиуса ядра. Сумма весов должна быть равна единице, поэтому после вычисления элементов ядра свёртки каждый делится на их сумму.

3.3.2. Математическая морфология

Простого применения бинарного порога к полученной разнице кадров иногда бывает недостаточно, так как на изображении остаётся шум. Например, это может быть вызвано, как уже отмечалось выше, движением листвы деревьев или воды в условиях ветра.

Чтобы избавиться от шума, часто используются операции бинарной математической морфологии. На вход принимается обрабатываемое бинарное изображение и так называемый структурный элемент (примитив), размер которого обычно много меньше изображения. Примитив зависит от операции, которая будет производиться над исходным изображением. Рассмотрим операции эрозии и наращивания на примере (Рис. 15).

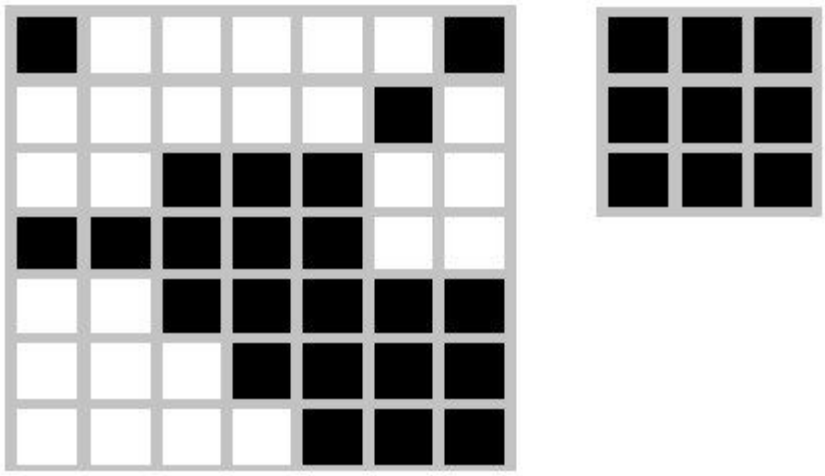

Рисунок 15. Обрабатываемое изображение (слева) и структурный элемент (справа)

Для обеих операций, как и в случае размытия по Гауссу, примитив применяется ко всем пикселям изображения. В случае наращивания структурный элемент будет полностью перенесён на изображение с последующим логическим сложением, если его якорь “наложился” на пиксель со значением 1. В случае эрозии же значение 1 останется только у пикселей, после “наложения” якоря структурного элемента на которые все элементы примитива со значением 1 соответствуют пикселям со значением 1. Если хотя бы одному элементу примитива со значением 1 соответствует пиксель со значением 0, то пикселю, на который был наложен якорь, будет присвоено значение 0 (Рис. 16).

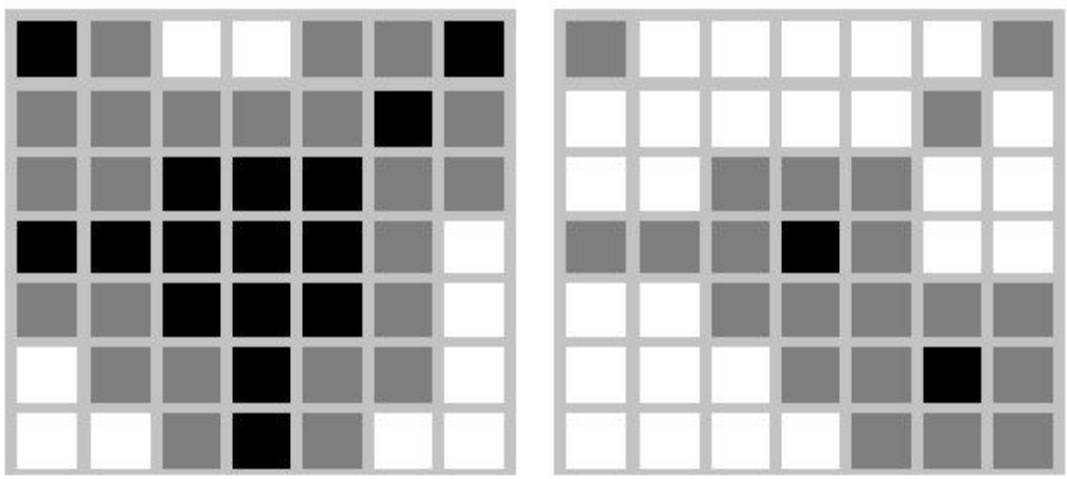

Рисунок 16. Результаты применения наращивания (слева) и эрозии (справа)

Математически операции наращивания и эрозии соответственно представляются следующими формулами:

$$A \oplus B = \cup_{b \in B} A_b$$

$$A \ominus B = \{z \in A | B_z \subseteq A\}$$

В этих формулах A *- бинарное изображение,* B *- структурный элемент*

Для удаления шума используют операцию размыкания, представляющую собой последовательное применение операции эрозии и операции наращивания с тем же структурным элементом. Наращивание производится для того, чтобы избежать побочного эффекта от эрозии в виде сильного уменьшения объектов (Рис. 17).

Рисунок 17. Процесс удаления шума с помощью размыкания

Глава 4. Выделение движущихся объектов

После того, как получена бинарная маска, характеризующая движение на изображении, с её помощью наконец могут быть выделены сами движущиеся объекты. Этот процесс выполняется в три шага:

1. Выделение контуров (внешних очертаний) полученных вычитанием фона предполагаемых объектов с помощью контурного анализа [26].
2. Получение прямоугольников, ограничивающих эти контуры.
3. Анализ найденных подобласти изображения.

Чтобы можно было заявить, что найдены именно движущиеся объекты, необходим третий шаг, ведь первые два будут гарантировать только нахождение областей движения. Алгоритмы вычитания фона крайне чувствительны к изменению освещения, поэтому изменение освещения на какой-либо части изображения (например, вызванное светом фар проезжающей машины, резко зашедшим за облака солнцем, светом из открывшейся двери и т.д.) может стать причиной ложных положительных результатов, т.е. влечь за собой ошибки второго рода (область, не являющаяся движущимся объектом, была принята за таковой). Кроме того, в случае, когда движущихся объектов несколько, возникает необходимость отличать их друг от друга, то есть необходимо осуществлять трекинг объекта. Именно по этим причинам необходима дополнительная информация о найденных областях.

Стоит отметить, что обычно в задачах трекинга либо имеется изображение-шаблон отслеживаемого объекта, либо вручную задаётся его начальное положение. Так как в рамках данной работы предполагается, что в общем случае не имеем ни каких-либо знаний об объекте, который будет осуществлять движение, ни его начального положения, то для определения

этих данных используются только информация об областях движения, полученных с помощью вычитания фона.

Далее будут предложены несколько подходов к анализу предполагаемых областей движения, а также будет проведён их сравнительный анализ.

4.1. Сопоставление особых точек

В настоящее время для сравнения двух изображений очень часто используют сопоставление особых точек [27, 28]. Этот подход можно применить и в рамках данной работы для сравнения областей движения, полученных на предыдущем и текущем кадрах.

Заметим, что так как перед построением гистограммы частот визуальных слов для каждого кадра уже были извлечены ключевые точки, достаточно просто найти те из них, которые принадлежат предполагаемой области движения. Сопоставление дескрипторов ключевых точек областей текущего (здесь и далее множество A) и предыдущего (множество B) кадров производится путём полного перебора (Brute-Force): для каждого дескриптора из множества A вычисляется расстояние до всех дескрипторов множества B (так как имеем бинарные дескрипторы ORB, то с помощью заданного выше расстояния Хэмминга), после чего выбирается наиболее близкий (Рис. 18).

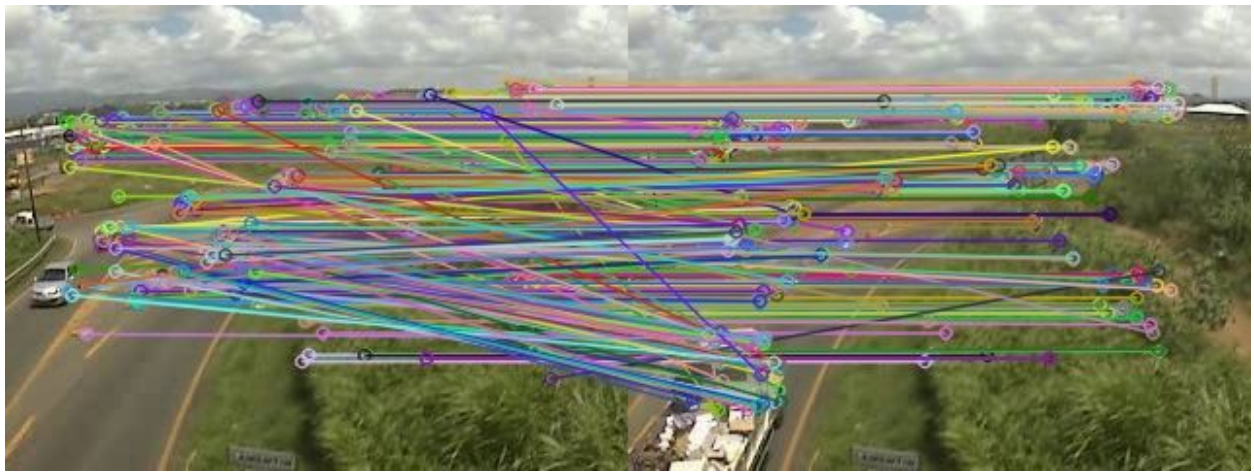

Рисунок 18. Сопоставление ключевых точек методом полного перебора

Когда мощность множеств A и B крайне высока, используется метод сопоставления дескрипторов, основанный на иерархических деревьях k-средних (в основе лежат алгоритмы библиотеки Fast Library for Approximate Nearest Neighbors - FLANN). В данной работе было решено его не применять, так как число дескрипторов в каждом множестве ограничено областью движения, размер которой предполагается значительно меньшим, чем размер кадра.

Рисунок 19. Самые лучшие соответствия после фильтрации

Не все особые точки сопоставляются корректно, и поэтому необходима их дополнительная фильтрация (Рис. 19). Далее будут рассмотрены фильтры, не требующих слишком больших вычислительных затрат.

4.1.1. Простые фильтры

Среди простых фильтров следует отметить следующие:

- Перекрёстная проверка.
- Проверка на соотношение.

Перекрёстная проверка подразумевает выполнение следующего условия: если для элемента a из множества A самым близким оказался элемент b из множества B, то и для b самым близким должен быть a. Такая фильтрация позволяет также избавиться от ситуаций, когда какой-либо ключевой точке ставится в соответствие сразу несколько особенностей, однако может отсечь и хорошие соответствия.

Для проверки на соотношение предполагается, что для элемента a из A имеется два наиболее близких элемента из B: b и c. Элементу a будет поставлен в соответствие b, если выполняется:

$$r_1 / r_2 > const$$

Здесь $const \in (0, 1)$, r_1 - расстояние между a и b, r_2 - между a и c. Если неравенство не выполняется, пара будет отброшена.

4.1.2. Итеративные фильтры. RANSAC

RANSAC [29] - итеративный метод получения параметров математической модели. Также существует множество его модификаций [30, 31], помогающих уменьшить время работы и увеличить качество. Метод позволяет осуществить фильтрацию посредством построения наилучшей матрицы преобразования особых точек (матрица гомографии в данном случае является моделью, для которой подбираются наилучшие параметры - её элементы). Вкратце описать этот процесс для одной итерации можно следующим образом:

1. Получение модели. Из элементов множества B случайным образом без повторений выбирается набор S_1 заранее заданного размера $|S|$. Из

множества A таким же образом выбирается набор S_2. Строится матрица преобразования элементов S_1 в элементы S_2.

2. Проверка модели. Для каждого элемента b из B, не задействованного для составления S_1, с помощью полученной матрицы строится его проекция. После этого для каждой полученной проекции выполняется поиск ближайшего элемента a во множестве A. Если расстояние между ними меньше некоторого порога, элемент b будет отнесён к выбросам (элементы, не соответствующие построенной модели). Остальные элементы будут использованы для последующих итераций.
3. Если построенная модель лучше предыдущей (количество выбросов меньше), происходит замещение модели.

Для фильтрации соответствий после построения матрицы гомографии для каждого элемента b из B достаточно построить его проекцию и проверить, достаточно ли она близка к соответствующей ему особой точке из множества A. Если нет, пара отбрасывается.

Даже после применения какого-либо из алгоритмов фильтрации могут остаться плохие связи, поэтому для достижения лучших соответствий на практике часто комбинируют эти методы. С этой целью в данной работе последовательно применяются перекрёстная проверка и RANSAC.

После фильтрации соответствий для решения, является ли объект на текущем кадре тем же объектом, что был найден на предыдущем, предлагается выбирать определённое количество K лучших из найденных соответствий, после чего осуществлять следующую проверку:

$$\frac{1}{K}\sum_{i=1}^{K} m_i < T$$

Где T *- заранее заданный порог (для бинарных дескрипторов ORB* $0 < T < 256$*),*

m_i - i-ое из K наилучших соответствий

4.2. Использование алгоритмов трекинга

Ещё одним подходом, рассмотренным в данной работе, является использование алгоритмов трекинга объекта, что позволит не полагаться на качество вычитания фона после того, как объект был найден и трекер был инициализирован. Чтобы в случае, когда объект начинает своё движение с границы кадра, произошёл “захват” всего объекта, а не на его части (Рис. 20), предлагается проводить инициализацию с запозданием на некоторое количество кадров d_{tr}. Также для того чтобы после того, как объект пропадает или перестаёт двигаться, переставать его отслеживать и продолжать поиск других движущихся объектов, предлагается задать максимальное количество кадров w_{tr} без движения отслеживаемого объекта. Это также поможет бороться с бликами и другими стационарными областями, ложно принятыми за движущийся объект. Первый подход в таких случаях будет допускать ошибки второго рода.

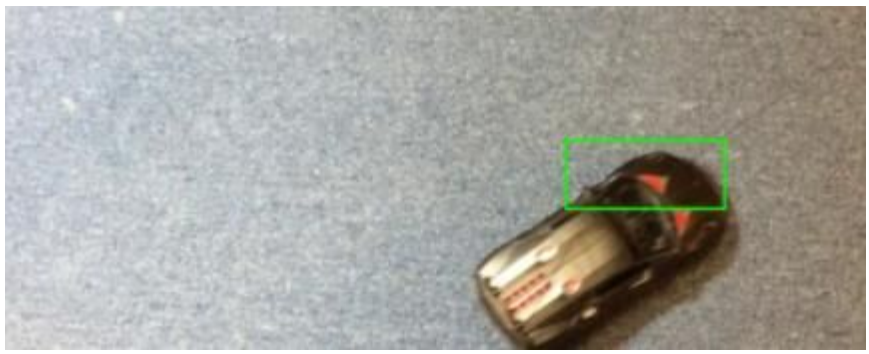

Рисунок 20. Пример плохой инициализации трекера (на части движущегося объекта)

Алгоритмы трекинга позволяют не осуществлять поиск объекта на каждом кадре, а строить его модель движения, если он уже был однажды найден, что обычно происходит во много раз быстрее. Более того, они дополнительно строят модель внешнего вида отслеживаемого объекта, которая позволяет им адаптироваться к изменению внешнего вида объекта

(например, в случае его вращения). Это также позволит продолжать отслеживать объект даже в случае его частичного перекрытия.

Одним из наиболее совершенных алгоритмов трекинга, широко использующимся на практике, является Kernelized Correlation Filter (KCF) [32], в основе которого лежат идеи регрессии с использованием функции ядра и свойства циркулярных матриц (Рис. 21). Из всех имеющихся на данный момент алгоритмов трекинга объектов был использован именно этот.

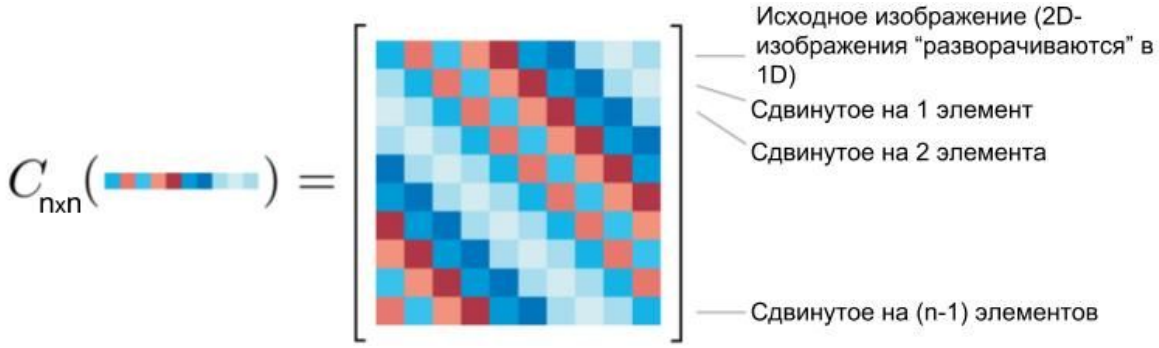

Рисунок 21. Построенная для изображения циркулярная матрица [32]

4.3. Оптический поток

Если совместить идеи первого и второго подходов, то для трекинга объекта в данной работе можно использовать оптический поток (Optical Flow), вычисляемый с помощью алгоритма Лукаса-Канаде [33]. Вычислять оптический вход для всего кадра - слишком медленно, поэтому на вход предлагается подавать координаты найденных с помощью ORB ключевых точек, принадлежащих области движения. Также как и в случае с использованием трекера, необходимо задать запоздание d_{tr}. Будем считать, что отслеживаем движущийся объект, если в области движения, получаемой с помощью вычитания фона, сохраняется 75% от изначального количества отслеживаемых вычислением оптического потока точек.

Основные идеи оптического потока следующие: рассматривается каждый пиксель изображения $I(x, y, t)$ в момент времени t и координатами

(x, y), который сместится на (dx, dy) на изображении, взятом через определённое время dt. Предполагая, что его интенсивность не изменится, можно записать:

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

В предположении, что перемещение маленькое, с помощью разложения в ряд Тейлора правой части уравнения получим:

$$I(x + dx, y + dy, t + dt) = I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt$$

Из этих двух равенств получаем:

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0$$

Или, поделив на dt и обозначив $f_x = \partial I/\partial x$, $f_y = \partial I/\partial y$, $f_t = \partial I/\partial t$ - градиенты изображения в соответствующих направлениях и $u = dx/dt$, $v = dy/dt$ - компоненты скорости, будем иметь так называемое уравнение оптического потока:

$$f_x u + f_y v + f_t = 0$$

Получаем уравнение с двумя неизвестными, которое можно решить с помощью метода Лукаса-Канаде: с использованием предположения, что оптический поток одинаков в локальной области и девяти точек, включая отслеживаемую. Получим переопределённую систему из 9-ти уравнений с 2-мя неизвестными, которая решается методом наименьших квадратов.

Подход будет хорошо находить только очень маленькие движения. Для того, чтобы он работал и для больших, крайне популярно использование пирамид Гаусса (Рис. 22).

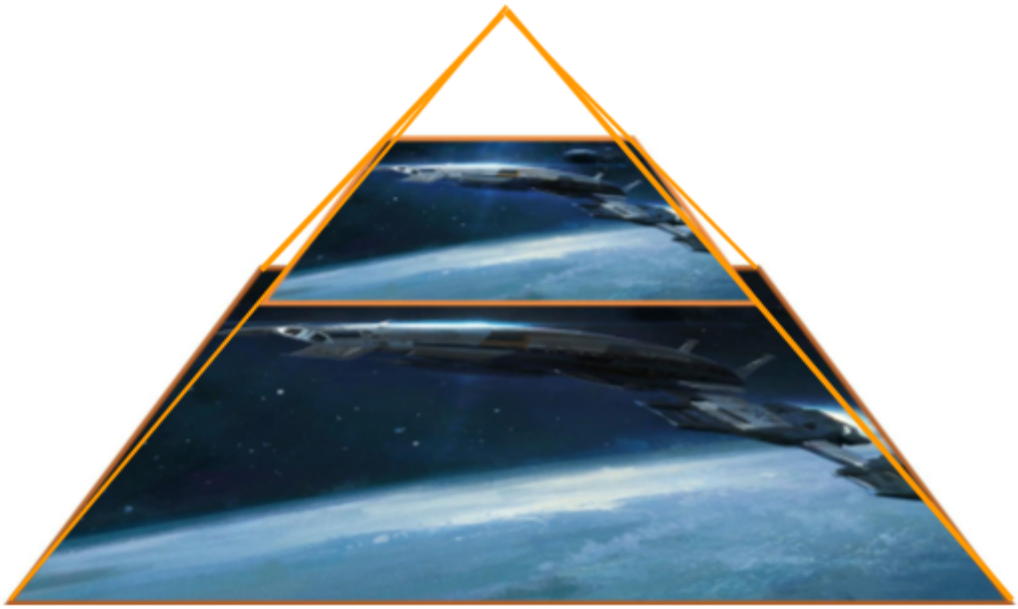

Рисунок 22. Гауссова пирамида

На верхнем уровне пирамиды большое для нижнего уровня движение будет хорошо отслеживаться, так как в этом случае оно будет малым, после чего найденная информация может быть использована для поиска движения на следующем уровне. Этот процесс повторяется итеративно, пока не дойдём до исходного изображения, соответствующего нижнему уровню пирамиды. Каждый последующий слой пирамиды получается следующим образом:

1. Размытие по Гауссу.
2. Прореживание изображения.

Такой способ позволяет осуществить построение пирамиды с минимальной потерей информации. Гауссовы пирамиды широко применяются в задачах обработки изображений, преследуя две цели:

- Ускорить процесс обработки изображений.
- По результатам обработки верхних уровней получить данные, которые помогут в обработке нижних (как в данном случае).

4.4. Сравнение подходов

В данном параграфе приведены результаты сравнения описанных выше подходов, основываясь на двух критериях: качеству получаемых результатов (визуальная оценка эксперта) и скорости работы, а также проведён их анализ. На основании этого сделаны выводы о применимости каждого из них.

4.4.1. Сравнение качества работы

Первый и третий подходы, основанные на ключевых точках, хорошо показали себя в ситуациях, когда находится много особых точек, принадлежащих объекту. К сожалению, ввиду плохого качества исходного изображения и малого размера движущегося объекта таких ключевых точек может оказаться мало или даже не оказаться вовсе. Более того, даже если попытаться для такой ситуации извлечь ключевые точки, но уже только для подобласти, содержащей объект, они могут найтись только в одном случае: если единственной причиной того, что ключевых точек не оказалось, послужило маленькое количество извлекаемых особенностей. Также для этого необходимы дополнительные вычислительные затраты.

К случаям, когда первый метод работает плохо, следует также отнести:

- Объект очень быстро вращается. Из-за этого могут очень сильно меняться его особенности, и разница между точками на текущем и предыдущем кадрах окажется слишком большой, что послужит причиной ошибок первого рода (движущийся объект не был принят за таковой).
- Если за область движения будет приниматься статичная область (например, из-за смены освещения на части изображения), она будет принята за движущийся объект, что послужит причиной ошибок второго рода.

Второй подход работает лучше остальных в тех случаях, когда имеется какая-то априорная информация об объектах, которые будут осуществлять движение, на основании которой можно выбрать подходящие d_{tr} и w_{tr}. Если этого сделать невозможно и эти параметры были подобраны неправильно, это наоборот очень сильно ухудшит результат.

Третий подход требует задания только одного параметра - d_{tr}, что несколько снижает уровень требований к априорной информации. Более того, когда для конкретного случая параметры заданы неправильно, он обеспечивает лучшие результаты, чем предыдущий.

4.4.2. Сравнение времени работы

В таблице 2 представлено среднее время обработки одного изображения для видео различных разрешений с использованием каждого из трёх рассмотренных в данной работе подходов.

Разрешение видео	**Подход 1 (Сопоставление ключевых точек)**	**Подход 2 (Использование трекинга)**	**Подход 3 (Оптический поток)**
1920 x 1080	0.18021	0.04626	0.10995
1280 x 720	0.08160	0.02062	0.04664
320 x 240	0.01242	0.00655	0.00889

Таблица 2. Среднее время обработки одного изображения (секунд)

Из приведённых результатов видно, что затраченное при использовании первого подхода время значительно превосходит время использования второго и третьего подходов. В то же время быстродействие второго гораздо выше, чем третьего.

4.4.3. Выводы

С учётом приведённых выше результатов можно отметить обратную зависимость между затрачиваемым подходами временем, необходимым для

вычислений, и требуемой ими для получения хорошего результата количеством априорной информации о предполагаемом объекте движения: чем больше подход требует априорной информации, тем быстрее он работает (Рис. 23).

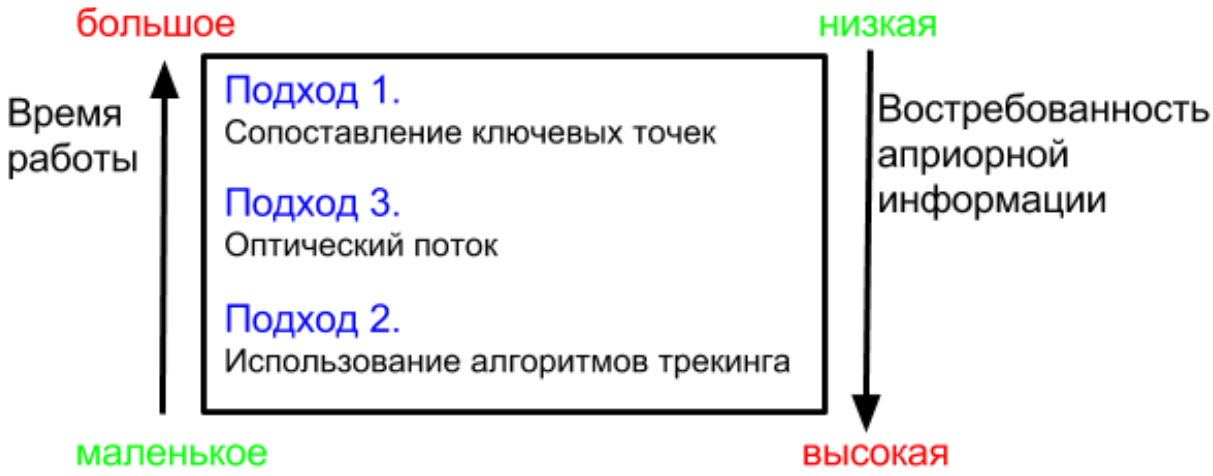

Рисунок 23. Зависимость между временем работы и востребованностью априорной информации

Также имеется прямая зависимость между количеством имеющихся данных о предполагаемом объекте движения и качеством работы программы, которого можно достичь, выбрав наиболее подходящий для конкретной ситуации подход (Рис. 24).

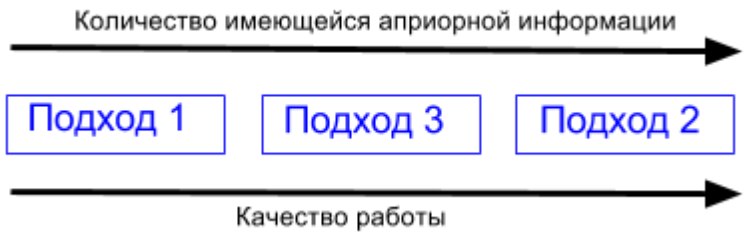

Рисунок 24. Зависимость между качеством работы и количеством априорной информации

Таким образом, чем больше имеется информации о предполагаемом объекте движения, тем более быстрый подход можно использовать, в то же время увеличив качество работы программы. Однако чем меньше этих знаний, тем хуже себя будут показывать быстрые подходы, и тогда для

сохранения приемлемого качества работы программы необходимо использование более медленных.

С учётом вышесказанного можно сделать вывод, что нельзя выделить какой-то определённый подход, универсальный для всех случаев. Для того, чтобы программа хорошо работала в любых условиях, было решено оставить все три с возможностью выбора одного из них с учётом конкретной ситуации.

Глава 5. Архитектура прототипа и его тестирование

В данной главе приведена архитектура спроектированного комплекса, схема взаимодействия его компонентов, а также проведено тестирование и дана предварительная оценка разработанного прототипа.

5.1. Архитектура решения

Для того чтобы программа хорошо работала даже при относительно низких вычислительных мощностях, было решено разделить её на две стадии работы (Рис. 25):

- Этап подготовки (офлайн вычисления). Сюда включены все основные вычисления.
- Этап работы в режиме реального времени (онлайн вычисления). Поиск похожего видео по кадру, поиск самого похожего кадра на данный в выбранном видео с учётом движения и построение траектории движущегося объекта.

Как можно заметить из предыдущих глав, алгоритмы для прототипа программы выбирались таким образом, чтобы максимально снизить время, затрачиваемое программой на втором этапе, то есть быстродействие второго этапа считалось приоритетнее, чем быстродействие первого. При этом было принято во внимание, что значительное увеличение требуемого на первом этапе времени также является недопустимым.

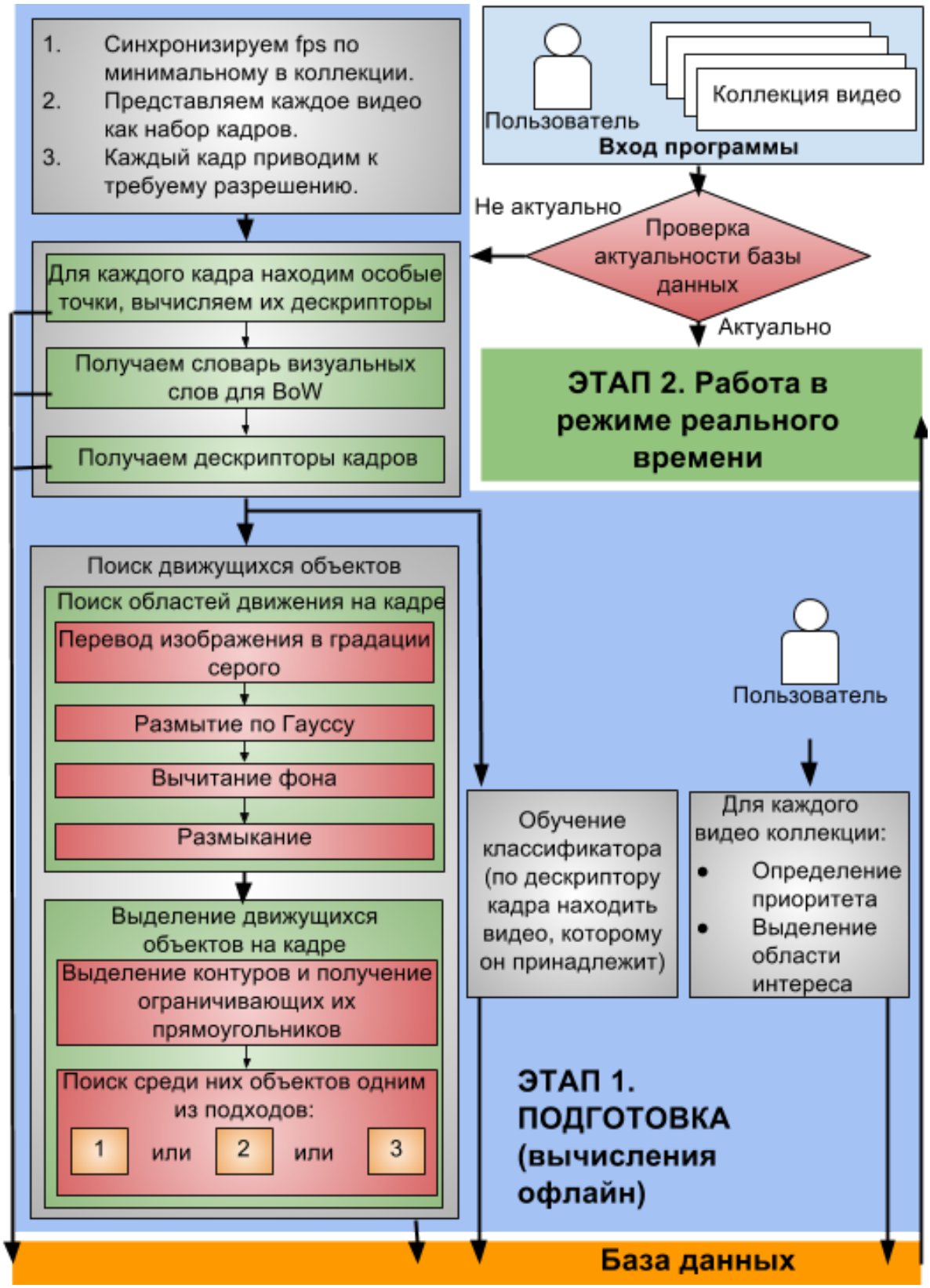

Рисунок 25. Общая схема программы, детальная схема этапа подготовки

5.1.1. Работа в режиме реального времени

Этап подготовки был подробно разобран в предыдущих главах, а его схема приведена на рисунке 25, поэтому отдельно разбирать этап офлайн вычислений в этом параграфе не будем. Вместо этого рассмотрим работу программы в режиме реального времени.

На этом этапе работы для построения траектории движущегося объекта используется кортеж C, содержащий n его местоположений. Для какого-либо конкретного кадра F местоположение добавляется к кортежу, если движущийся объект был найден на нём во время этапа подготовки. Если же на кадре F объект на этапе подготовки найден не был, из кортежа C удаляются k самых старых его элементов. Для построения траектории используется метод наименьших квадратов.

Работа программы в режиме реального времени организована следующим образом (Рис. 26): если на *текущем кадре* найден движущийся объект, то при условии, что $|C| = n$, строится его траектория. Если направление траектории совпадает с направлением, в котором находится область интереса (2 направления для оси x: вправо и влево, 2 направления для оси y: вверх и вниз), происходит её отображение.

Если траектория проходит через область интереса, предсказывается необходимое объекту количество кадров, чтобы её достигнуть, по формуле:

$$s_1 = speed \,/\, dist$$

Где $dist$ - расстояние от центра объекта до центра области интереса, а $speed$ - скорость объекта. Для каждой из координат x, y она вычисляется на основании имеющихся в C местоположений:

$$speed = \frac{1}{n} | c_1 - c_2 |$$

Где c_1, c_2 - первое и последнее местоположения в C

При этом для каждого кадра продолжает вычисляться траектория. После того, как объект достигает области интереса, происходит

переключение, чтобы избежать перекрытия (с этой же целью можно переключаться немного раньше, чтобы в случае, если переключение было предсказано значительно позже, чем должно произойти, избежать его полного отсутствия). Оно также происходит в случае, если на этапе подготовки объект был найден на s_2 подряд идущих кадрах. Если s_1 было вычислено и ожидается переключение, но в какой-то момент траектория изменилась настолько, что она больше не проходит через область интереса, то оно сбрасывается, то есть переключения не произойдёт.

Поиск кадра, на который следует переключиться, происходит следующим образом:

1. С помощью классификатора предсказываются вероятности принадлежности p_i текущего кадра каждому видео V_i, где $i \in [1, V]$ - номер видео, V - количество видео в коллекции.
2. Для каждого видео вычисляется число $d_i = pr_i * p_i$, где pr_i - приоритет видео. Максимальное из них (не учитывая вычисленного для текущего видео) будет соответствовать видео, на которое следует переключиться.
3. Среди этого видео находим только те кадры, в которых на этапе подготовки был найден движущийся объект, и сравниваем их гистограммы с гистограммой текущего кадра. Переход будет осуществляться в наиболее близкий кадр. Для вычисления расстояния использовалось ядро хи-квадрат (1).

Рисунок 26. Схема работы программы в режиме реального времени

5.1.2. Индексирование

Чтобы согласовать работу программы на этапе подготовки данных и на этапе работы в режиме реального времени, а также чтобы увеличить быстродействие второго, использовался специальный подход к хранению данных (Рис. 27). Как видно из рисунка 25, полученные технические данные на этапе подготовки сохранялись, чтобы можно было их использовать повторно, избежав лишних вычислений.

Рисунок 27. Структура базы данных

Помимо представленных на рисунке 27 таблиц для хранения данных также использовались следующие дополнительные файлы:

- Содержащий все имеющиеся дескрипторы кадров. Поиск нужного по полю “ID” таблицы “кадры”.
- Содержащий все имеющиеся дескрипторы особых точек. Поиск нужного по полю “ID” таблицы “особые точки”

5.1.3. Данные, вводимые пользователем

Как видно из рисунков 25 и 26, пользователь:

1. Проверяет актуальность имеющихся данных и в случае, если они не актуальны, запускает процесс их подготовки.

2. Устанавливает приоритет видео и область интереса для каждого из них.
3. Выбирает начальное видео и кадр для начала работы программы в режиме реального времени.

Для второго (Рис. 28) и третьего пунктов были разработаны графические интерфейсы пользователя (Graphical User Interface, GUI) на основе библиотеки OpenCV.

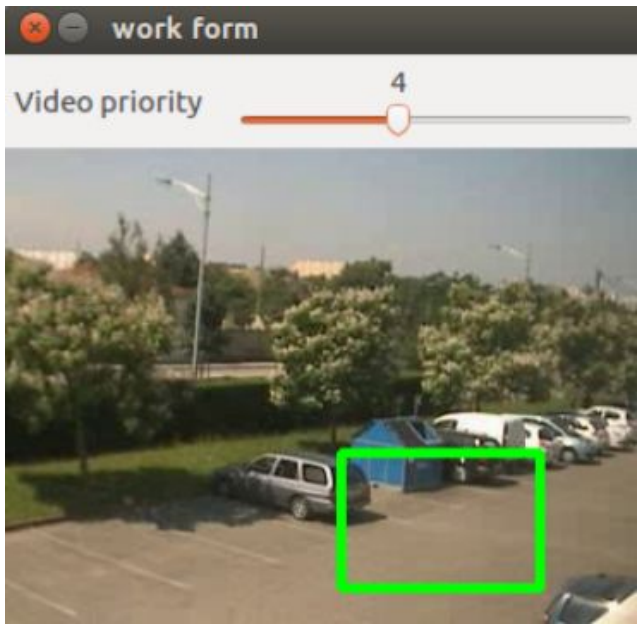

Рисунок 28. GUI для выбора приоритета и области интереса. Пример для одного из видео

5.2. Тестирование прототипа

Для тестирования разработанного прототипа использовался тестовый стенд со следующей конфигурацией:

- Процессор Intel Core i7-3630QM (2.40 GHz).
- Оперативная память 8 Gb DDR3 (1600 MHz).
- Жёсткий диск (5400 RPM, SATA III, 6 Gb/s).

Имплементация прототипа программы, основанной на разработанной архитектуре, была протестирована при различных входных параметрах на двух различных коллекциях (Рис. 29). Синий прямоугольник отображает положение области интереса, красный - движущийся объект, красные точки - положения центров движущегося объекта, использующиеся для построения

траектории. Траектория изображена стрелкой. Красная стрелка - объект перекроет область интереса, фиолетовая - объект движется в сторону области интереса.

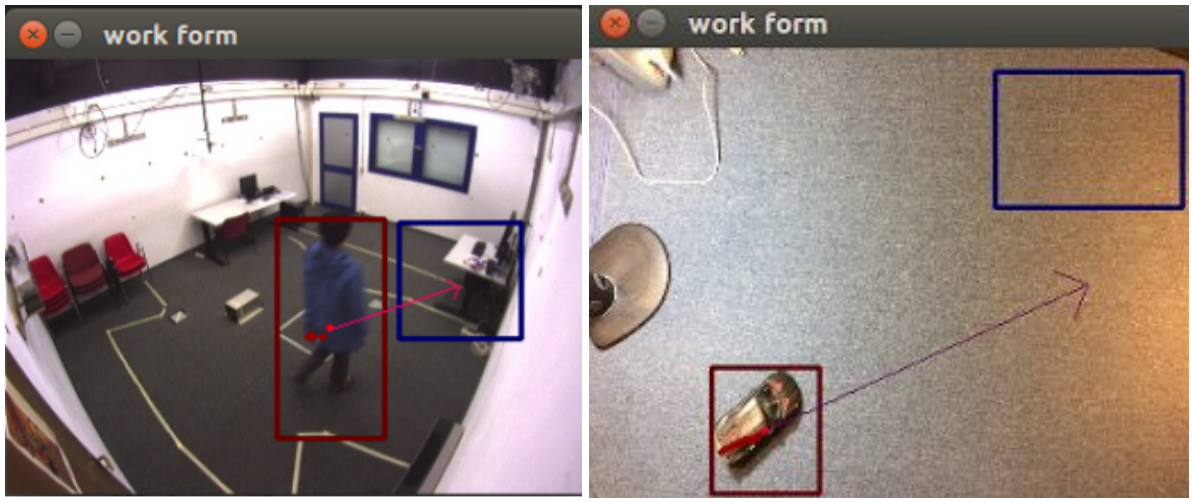

Рисунок 29. Пример работы программы

Для разработки прототипа использовались следующие инструменты:

- Операционная система Ubuntu 16.04.
- Язык программирования Python 3.5.2.
- Библиотека OpenCV 3.2.0.
- Библиотека SQLite 3.11.0.
- Библиотека Scikit-learn 0.18.1.

Первая коллекция содержит 3 видео, средняя длина которых составляет 200 кадров, средняя продолжительность - 40 секунд. Вторая содержит 4 видео, полученных с установленных по углам квадратной комнаты камер, каждое из которых содержит 400 кадров. Продолжительность - около трёх с половиной минут. В таблицах 3 и 4 представлены полученные результаты времени работы программы для первой и второй коллекции соответственно. При этом используются следующие обозначения:

- S - размер словаря BoW
- F - количество извлекаемых из кадра особых точек
- res - разрешение кадров после применения компрессии.

- R - среднее время работы для одного кадра в режиме реального времени.
- P - время, требуемое на этап подготовки.

(S, F, res)	R, сек.	P, сек.
20, 10, 320x240	**0.0037**	**58**
100, 200, 320x240	0.0038	1581
20, 10, 1980x1080	0.045	327

Таблица 3. Время работы программы для первой тестовой коллекции (секунд)

(S, F, res)	R, сек.	P, сек.
20, 10, 320x240	**0.00282**	**78**
100, 50, 320x240	0.00284	1479
20, 10, 1024x768	0.015	323

Таблица 4. Время работы программы для второй тестовой коллекции (секунд)

5.2.1. Результаты профилирования второго этапа

Архитектура программы разрабатывалась таким образом, чтобы R не зависело от количества извлекаемых из кадра особенностей и его разрешения, при этом сохраняя слабую зависимость от S, длины видео в коллекции и их количества. Однако при тестировании прототипа выяснилось, что существует крайне высокая зависимость от разрешения кадров. Для того, чтобы выяснить причину, было проведено профилирование с помощью встроенного в Python 3.5.2 модуля profile, результаты которого представлены в таблице 5.

Функция	Время выполнения на один вызов (сек.)	% от общего времени выполнения программы
cv2.imread	0.017	83.4
cv2.imshow	0.003	12.9
cv2.rectangle	< 0.001	0.43

Таблица 5. Результаты профилирования. Первые три функции, требующие наибольшего количества времени на вычисления (по убыванию)

Из результатов профилирования видно, что функции библиотеки OpenCV imread и imshow, отвечающие за получение изображения и его отображение пользователю соответственно, работают достаточно медленно, что и является причиной снижения скорости работы программы в режиме реального времени для кадров высокого разрешения.

5.2.2. Анализ результатов тестирования и выводы

Таким образом, было проведено тестирование, показавшее высокую требовательность к вычислительным ресурсам некоторых процессов (например, отображение и чтение изображений на обоих этапах работы программы, а также кластеризация дескрипторов кадра на этапе подготовки), из чего можно сделать вывод, что в дальнейшем предстоит провести много работы по их оптимизации.

Хотя прототип и реализовал все поставленные задачи, не лучшим образом работает основанный на дескрипторах BoW поиск кадра в найденном видео, на который будет осуществлён переход, причём результат не зависит ни от разрешения видео, ни от количества извлекаемых из изображений особенностей и размера словаря BoW. Это говорит о том, что в дальнейшем предстоит:

- Провести исследование зависимости между качеством сравнения дескрипторов кадров и вектором входных параметров (S, F, res).
- Провести общее исследование, направленное на поиск метода нахождения в видео кадра, на который необходимо произвести переход, по некоторому принятому на вход изображению.

Также стоит отметить, что необходимость в этой части программы вовсе отпадает, если ослабить ограничения на видео в коллекции (положить, что не все операторы обязательно принадлежат сцене), при этом потребовав полную синхронизацию по времени принадлежащих рассматриваемой сцене видеопотоков.

Все остальные этапы работы программы показали себя очень хорошо, и, что самое важное, полученный результат оставался на крайне высоком уровне даже при невысоких значениях вектора входных параметров (выделены жирным в таблицах), что позволяет использовать разработанную программу даже при небольших вычислительных мощностях. В дальнейшем результат можно улучшить с использованием графических процессоров (GPU), распараллеливания и доработки реализации входящих в систему элементов (в частности, с помощью переноса прототипа с изначально выбранного для прототипирования языка программирования Python на C++). Также планируется осуществить плавные переходы между участками видеопотока либо используя матрицу гомографии для склейки изображений, либо с помощью 3D-визуализации сцены.

Заключение

Таким образом, в рамках представленной работы:

- После изучения публикаций и существующих решений была спроектирована архитектура комплекса, позволяющего объединять технологии построения виртуального видеопотока и видеоаналитики. При этом был проведён анализ следующих методов:
  - сравнения изображений и видео;
  - поиска областей движения;
  - выделения движущихся объектов.
- Был проведён эксперимент, который позволил полностью подтвердить выдвинутую гипотезу, установив, что существуют средства и методы, позволяющие построить виртуальный видеопоток с использованием данных о движущихся объектах без высоких затрат на приобретение дорогостоящего оборудования и программного обеспечения. В ходе эксперимента для уменьшения времени работы программы в режиме реального времени формировались и использовались файлы со специальной структурой.
- В результате эксперимента было получено решение, представленное в виде прототипа программного обеспечения. Ссылку на репозиторий можно найти в приложении.

Всё вышеперечисленное позволило существенно приблизиться к поставленной цели, и, что самое главное, получить бесценный опыт решения задач машинного обучения и компьютерного зрения, а также проектирования и разработки программного обеспечения.

Список литературы и источников

1. Продукты Kipod // Synesis URL: http://synesis.ru/products (дата обращения: 12.01.2017)
2. Производство прямых трансляций // Sony URL: http://www.sony.ru/pro/products/solutions-live-production (дата обращения: 11.01.2017).
3. B&H Photo Video Digital Cameras, Photography, Camcorders URL: http://www.bhphotovideo.com (дата обращения: 04.04.2016).
4. WTS Broadcast: Broadcast and Production URL: http://shop.wtsbroadcast.com (дата обращения: 04.04.2016).
5. LensFrame (Video Stitching Software) // Coherent Synchro URL: http://www.coherentsynchro.com/portfolios/seo-optimization/ (дата обращения: 12.01.2017).
6. Джгаркава Г.М., Лавров Д.Н. Использование метода Surf для обнаружения устойчивых признаков изображения при создании сферических панорамных снимков // Математические Структуры и Моделирование, №1 (22), С. 96-99, 2011.
7. E. Rublee, V. Rabaud, K. Konolige, G. Bradski. ORB: an efficient alternative to SIFT or SURF // Proceedings of 13th International Conference on Computer Vision, pp. 2564-2571 , 2011.
8. E. Rosten, T. Drummond. Machine learning for high speed corner detection // 9th European Conference on Computer Vision, Vol. 1, pp. 430–443, 2006.
9. M. Calonder, V. Lepetit, C. Strecha, P. Fua. BRIEF: Binary Robust Independent Elementary Features // 11th European Conference on Computer Vision (ECCV), Heraklion, Crete. LNCS Springer, 2010.
10. Волков А.Г., Семёнов С.В., Севрюков С.Ю. Оценка влияния размера словаря «Мешка визуальных слов» и компрессии кадра на скорость и

точность классификации видео // Процессы управления и устойчивость, Т. 3 (19), С. 362-366, 2016.

11. J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations // Proc. Fifth Berkeley Symp. Math. Statistics and Probability, vol. 1, pp. 281-296, 1967.
12. E. Forgey. Cluster Analysis of Multivariate Data: Efficiency vs. Interpretability of Classification // Biometrics, vol. 21, p. 768, 1965.
13. Costantino Grana, Daniele Borghesani, Marco Manfredi, Rita Cucchiara. A fast approach for integrating orb descriptors in the bag of words model // Proc. of IS&T/SPIE Electronic Imaging: Multimedia Content Access: Algorithms and Systems, vol. 8667, pp. 091-098, San Francisco, California, US, Feb 2013.
14. C. Cortes, V. Vapnik. Support-vector networks // Machine Learning, vol. 20(3), pp. 273-297, 1995.
15. J. Zhang, M. Marszalek, S. Lazebnik, C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study // International Journal of Computer Vision, vol. 73(2), pp. 213-238, 2007.
16. T.-F. Wu, C.-J. Lin, R. C. Weng. Probability estimates for multiclass classification by pairwise coupling // Journal of Machine Learning Research, vol. 5, pp. 975-1005, 2004.
17. M.E. Tipping, A.C. Faul. Fast marginal likelihood maximisation for sparse Bayesian models // In C. M. Bishop and B. J. Frey (Eds.), Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, Key West, FL, Jan 3-6, 2003.
18. V. Ayumi, M.I. Fanany. A Comparison of SVM and RVM for Human Action Recognition // Internetworking Indonesia Journal, vol. 8, No. 1, pp. 29-33, 2016.

19. M. Rafi and M. S. Shaikh. A comparison of SVM and RVM for Document Classification // Procedia Computer Science, vol. 00, pp. 3-8, 2013.
20. S. Jeeva, M. Sivabalakrishnan. Survey on Background Modeling and Foreground Detection for Real Time Video Surveillance // Procedia Computer Science, vol. 50, pp. 566-571, 2015.
21. R. Collins, A. Lipton, T. Kanade, H. Fijiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, L. Wixson. A system for video surveillance and monitoring // Proceedings of the IEEE, Vol. 89, pp. 1456-1477, 2001.
22. M. Van Droogenbroeck, O. Barnich. Vibe: A disruptive method for background subtraction. // In T. Bouwmans, F. Porikli, B. Hoferlin, A. Vacavant, editors, Background Modeling and Foreground Detection for Video Surveillance, chapter 7. Chapman and Hall/CRC, pages 7.1-7.23, July 2014.
23. P. Kaewtrakulpong, R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection // Video-Based Surveillance Systems, pp. 135-144. Springer, 2002.
24. Z. Zivkovic, F. Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction // Pattern recognition letters, Vol. 27(7), pp. 773-780, 2006.
25. L. Shapiro, G. Stockman. Computer Vision // Prentice Hall, 2001.
26. S. Suzuki, K. Abe. Topological Structural Analysis of Digitized Binary Images by Border Following // Computer Vision, Graphics, And Image Processing, Vol. 30, pp. 32-46, 1985.
27. S. Gauglitz, T. Hollerer, M. Turk. Evaluation of Interest Point Detectors and Feature Descriptors for Visual Tracking // International Journal of Computer Vision, Vol. 94, pp. 335-360, 2011.

28. L. Yu , Z. Yu, Y. Gong. An Improved ORB Algorithm of Extracting and Matching Features // International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol. 8, No. 5, pp. 117-126, 2015.
29. M. Fischler, R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography // Communications of the ACM, vol. 24, pp. 381–395, 1981.
30. O. Chum, J. Matas. Matching with PROSAC - Progressive Sample Consensus // Proceedings of Conference on Computer Vision and Pattern Recognition, Vol. 1, pp. 220-226, 2005.
31. R. Raguram, JM. Frahm, M. Pollefeys. A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus // D. Forsyth, P. Torr, and A. Zisserman (Eds.): ECCV 2008, Part II, LNCS 5303, pp. 500-513, 2008.
32. J. Henriques, R. Caseiro, P. Martins, J. Batista. High-Speed Tracking with Kernelized Correlation Filters // IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 37, No. 3, pp. 583-596, 2014.
33. JY. Bouguet. Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker Description of the Algorithm // Intel Corporation, Vol. 5, pp. 1-10, 2001.

Приложение. Рекомендуемые электронные ресурсы

1. Одна из используемых в работе тестовых коллекций видео: https://tev-static.fbk.eu/DATABASES/MVPDT.html
2. Используемая в работе имплементация классификатора RVM: https://github.com/AmazaspShumik/sklearn-bayes
3. Отслеживание движения и алгоритмы сопровождения ключевых точек: http://www.intuit.ru/studies/courses/10622/1106/lecture/18022
4. SVM (SKLearn): http://scikit-learn.org/stable/modules/svm.html
5. Размытие по Гауссу: http://www.pixelstech.net/article/1353768112-Gaussian-Blur-Algorithm
6. Математическая морфология: https://habrahabr.ru/post/113626/
7. Анализ алгоритмов трекинга из OpenCV: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/
8. Контурный анализ в OpenCV: http://robocraft.ru/blog/computervision/640.html
9. Документация OpenCV:
   http://docs.opencv.org/trunk/d1/d89/tutorial_py_orb.html
   http://docs.opencv.org/trunk/db/d5c/tutorial_py_bg_subtraction.html
   http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html
   http://docs.opencv.org/3.2.0/d7/d8b/tutorial_py_lucas_kanade.html
   http://docs.opencv.org/3.2.0/db/df6/tutorial_erosion_dilatation.html
10. Профилировщик Python: https://docs.python.org/2/library/profile.html
11. Репозиторий с прототипом программы: https://github.com/DEARTSV/virtual_videostream